SYMBOLIC ALGEBRAIC MANIPULATION BY DIGITAL COMPUTER IN PROBLEMS OF CONTROL THEORY

Klaus Peter Merzhäuser

United States
Naval Postgraduate School

THESIS

SYMBOLIC ALGEBRAIC MANIPULATION
BY DIGITAL COMPUTER
IN PROBLEMS OF CONTROL THEORY

by

Klaus Peter Merzhäuser

June 1970

Symbolic Algebraic Manipulation By Digital Computer In Problems of Control Theory

by

Klaus Peter Merzhäuser
Lieutenant Commander, Federal German Navy
Ingenieur für Elektrotechnik, SIS Wuppertal 1960

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 1970

ABSTRACT

A method of digital computer oriented analysis of a Mason flow graph is presented. Using this method in connection with a computer program implementing the algebraic manipulation language FORMAC can be a tool in expanding large determinants with symbolic polynomial elements and thereby finding the characteristic equation and transfer functions of a linear time-invariant system in symbolic form.

Also a new approach to the root locus method is shown, using a FORMAC program. The advantages over the conventional root locus method are discussed.

Areas of possible future use of FORMAC in algebraic problems of control theory are discussed.

TABLE OF CONTENTS

I. INTRODUCTION ---------- 7

A. THE NATURE OF THE PROBLEM ---------- 7

B. METHOD OF SOLUTION ---------- 7

C. THE ALGEBRAIC MANIPULATION LANGUAGE "FORMAC" --- 10

II. METHOD OF COMPUTER ANALYSIS FOR A MASON FLOW GRAPH ---------- 13

A. DEFINITIONS ---------- 13

1. The Mason Flow Graph ---------- 13

2. Input- and Output-Nodes, Branchpoint Node -- 14

3. Input List LIST1 ---------- 14

4. LIST2 and LIST3 ---------- 16

5. Topological Matrices ---------- 16

a. Loop Matrix ---------- 16

b. Path Matrix P ---------- 18

c. Touching Loops Matrix LT ---------- 18

d. Path - Loop Touching Matrix PL -------- 18

e. Control Vectors ---------- 18

(1) Loop Search Control Vector LC ---- 19

(2) Multiple Node Control Vector LIST1(4) ---------- 19

(3) Branchpoint Control Vector BC ---- 19

6. Logical Operations On Matrices ---------- 19

a. The Operation ⊗ ---------- 19

B. THE LOOP SEARCH ---------- 20

1. Initialization ---------- 20

2. Search For Loops ---------- 22

a. Finding Successive Nodes ---------- 22

b. Detection Of a Loop ---------- 23

c. Recording Of the Loop ---------- 23

d. Completion Of the Loop Search --------- 24

C. THE PATH SEARCH ---------- 25

1. Initialization ---------- 25

2. Path Search ---------- 26

a. Detection Of a Path ---------- 26

b. Recording Of the Path ---------- 28

D. EVALUATION OF TRANSFER FUNCTIONS ---------- 28

1. Evaluation of the Graph Determinant Δ (DELTA) ---------- 29

a. Evaluation Of the Matrix LT ---------- 29

b. Evaluation Of DELTA By a FORMAC Program ---------- 31

2. Evaluation Of the k^{th} Cofactor Δ_k (DELTK) -- 31

3. Evaluation Of the k^{th} Path Gain (PK) ------- 33

4. Final Evaluation Of the Transmission Gain -- 37

III. DIRECT EVALUATION OF DETERMINANTS USING FORMAC ----- 39

A. A FORMAC PROGRAM USING THE BASIC DEFINITION OF THE DETERMINANT ---------- 39

1. General Description Of the Evaluation Of the Determinant By the FORMAC Program DETERM --- 41

a. The Non-Algebraic Part Of DETERM ------ 41

b. The Algebraic Part Of DETERM ---------- 42

2. Discussion Of the Flow Graph For DETERM ---- 50

a. Data Design For DTERM ---------- 50

b. The Logic Sequence ---------- 56

B. EVALUATION OF A DETERMINANT USING BARREISS FRACTION-FREE ALGORITHM ---------- 56

1. The Fraction-Free Gaussian Elimination ----- 57

IV. APPLICATION OF FORMAC IN OTHER ALGEBRAIC PROBLEMS OF CONTROL THEORY ---------- 59

A. ROOT LOCUS THEORY ---------- 59

1. A FORMAC Program For the Root Locus -------- 60

2. Discussion ---------- 61

B. FREQUENCY FUNCTIONS ---------- 62

C. LAPLACE TRANSFORMS AND INVERSION OF LAPLACE TRANSFORMS ---------- 62

D. SENSITIVITY ANALYSIS ---------- 63

V. CONCLUSION ---------- 65

COMPUTER PROGRAMS ---------- 66

BIBLIOGRAPHY ---------- 92

INITIAL DISTRIBUTION LIST ---------- 93

FORM DD 1473 ---------- 95

I. INTRODUCTION

A. THE NATURE OF THE PROBLEM

Consider a linear time-invariant system whose parameters are known only symbolically. In the analysis of the system often it becomes necessary to derive certain system functions which are polynomials in the Laplace variable s.

The "manual" derivation of these functions, even for small systems, becomes a task which is not only time consuming but unattractive, and although conceptually simple, is also a source of errors which are difficult to detect.

The enormous work involved in the deriving of these functions for medium or large systems can be the reason for not attempting a solution to the problem in general algebraic form.

There are also possible other algebraic problems where the "manual" derivation is tedious.

The problem, therefore, is to devise methods of using the digital computer to perform those algebraic manipulations.

B. METHOD OF SOLUTION

The first specific problem to be considered will be the evaluation of the characteristic equation of a linear system.

To solve this problem it will be necessary to evaluate the determinant of a square matrix whose elements are generally polynomials in the variable s, derived from the differential equations of the system by Laplace transform.

Two different methods of evaluating a determinant are applied here. The first method is based on the theory of

graphs, specifically on the properties of a Mason flow graph (1). It will be assumed that the reader is familiar with the Mason gain formula.

Other graph methods are those of Coates and the modified method of Chan and Bapna [Ref. 2]. These methods are not based on a Mason flow graph. A flow graph analysis using a Mason flow graph and based on a topological development is given by Dunn and Chan [Ref. 3].

The flow graph analysis used here is not pure topological, because the loops and paths are not found from the topological properties of the graph but by search methods.

A comparison of flow graph techniques and another different method by Chan and Mai are found in [Ref. 2].

The basic feature of the mechanized graph analysis used here is the process of identifying closed unidirectional paths called "loops" among the connections of the graph. Then the set of non-touching loops is found which is then used to evaluate the determinant.

The second method used here for the evaluation of a determinant is using a fraction-free algorithm. Lipson [Ref. 4] uses this algorithm in the evaluation of a set of linear algebraic equations.

The basic problem in the evaluation of the determinant Det(A) is the identification of those elements in the square matrix A whose products form a term in the determinant. There are n! terms in the determinant of a square matrix of order

n; most of them are often equal to zero in a linear system of the kind considered in control theory.

The computer program DTERM (Program 1) shows in its algebraic part how products of polynomials can be processed without using excessive space (which was the main limitation imposed).

The first, non-algebraic part of DTERM uses the basic definition of a determinant to find the non-zero products in the determinant by forming all possible permutations of the column indices of the elements of the matrix A. This method was used only to simulate the presence of a result of the flow graph analysis, which was not programmed.

For matrices of order 5 it was found that the execution time for investigation of all 120 possible products of 5 elements was 6 sec, for order 8 the execution time was 27 sec and for order 9 the execution time was 3 min 43 sec. These times include also other statements.

The fraction-free algorithm is given here as a different possible method to evaluate the determinant of a matrix, using only single letters as symbolic elements followed by a gradual replacement by the longer polynomials. The latter process is the problematic one because of the enormous space requirement which can be generated. It would probably be similar to the method used in the algebraic part of DTERM.

For the other algebraic problems mentioned here short example programs in FORMAC, an algebraic manipulation language,

are given. These programs show the possible use of FORMAC in the solution of those problems; they are not optimal.

C. THE ALGEBRAIC MANIPULATION LANGUAGE "FORMAC"

The algebraic manipulation must be performed with the aid of a special tool: an algebraic manipulation language. Here "FORMAC" was used for this purpose.

FORMAC stands for FORmula MAnipulation Compiler. It was developed by the IBM Corporation as "an experimental system to provide engineers with the capability of handling formal mathematical expressions on a computer"[5].

Examples of application of FORMAC are given in [6] and [7]. Reference to bibliographies is made in [8].

The most recent form of FORMAC is available in the PL1-Version. The language PL/1 is a proper subset of FORMAC, it is therefore possible to use the entire PL/1 capability together with FORMAC. The basic reference manual for PL/1 [9] together with the respective manual for FORMAC [6] at this time (1970) is all the user must be referred to. The average FORTRAN user can however in a few days acquire effective knowledge of both PL/1 and FORMAC in order to communicate with the FORMAC language.

FORMAC enables the user to analyze and synthesize general algebraic expressions.

The work with FORMAC proceeds basically in such a way that the user writes a program in the FORMAC language, following the route of analysis of normal "manual" algebraic manipulation.

The capabilities of FORMAC can be divided into the following groups:

- analysis: test for mathematical identity, finding the coefficient of an expression inside a parent expression, finding the highest or lowest power of one expression inside a parent expression, finding the numerator or denominator of a rational expression, finding the main operator of an expression (+,-,:,x, and so on), finding the number of terms of an expression (factors or summands), selection of a specified term of an expression.

-synthesis: assignment of an algebraic expression to a FORMAC variable, user defined function processing, conversion routines, well developed transit between "normal" values and PL/1 variables and FORMAC variables, multiplication, division, addition and subtraction, differentiation with respect to specified and unspecified arguments, use of trigonometric and natural functions with numeric and symbolic arguments, combinatorial and factorial integer functions.

-support: simple printout, editing, control over level of expansion and evaluation.

II. METHOD OF COMPUTER ANALYSIS FOR A MASON FLOW GRAPH

In this chapter a procedure will be described which can be implemented using a digital computer. The basic operations performed by the computer are list searches and logic operations on binary matrices.

The method produces:

- lists of node numbers which form a specific loop.
- lists of associated branch gains, so loop gains can be evaluated.
- a list containing the names of touching loops, so sets of touching loops can be identified.
- lists of node numbers which form a path from a specified node to another specified node. All the lists identify all the paths present between these nodes.
- lists of associated branch gains, so the different path gains can be evaluated.
- a list of branch point nodes which identify the points where the different paths branch.

First the terms used will be defined, then the method will be described, using examples. The computer implementation will be discussed.

A. DEFINITIONS

1. The Mason Flow Graph

Here the definitions and notation of Ward and Strum[1] will be used.

The Mason flow graph consists of a set of nodes, which are here arbitrarily numbered, and a set of branches, which have a direction and a weight associated with them. The branches connect the nodes in such a way that at least every node is connected with one other node; self loops are not present.

2. Input- and Output-Nodes, Branchpoint Node

An input node has no incoming branches. Here an output node has one incoming branch. A branchpoint node is a node with at least two outgoing branches. From a branchpoint node there are at least two paths leading to a common end node.

End- and starting nodes are the end- and the starting point respectively of paths.

3. Input List LIST1

The input data are prepared in the form of an input list, which is called "LIST1". Each item in LIST1 has three parts: two node numbers and a branch designator.

These definitions are illustrated in Fig. 1 and 2.

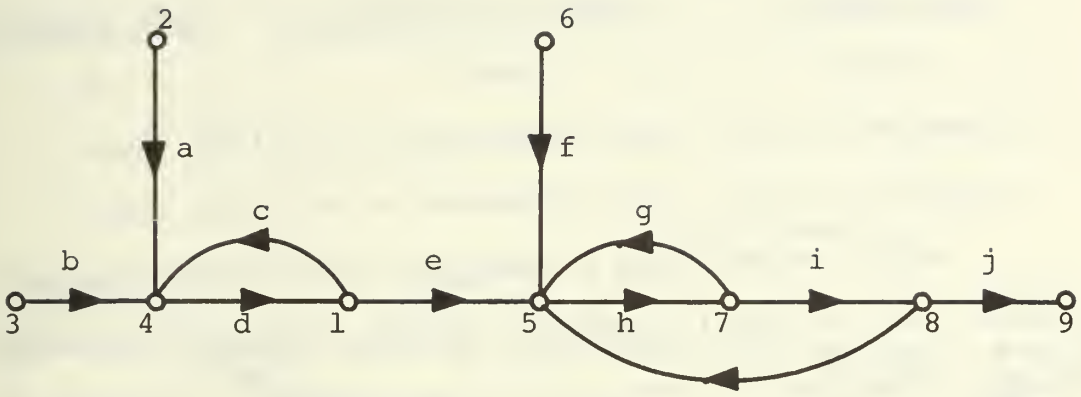


Fig. 1

Nodes 3, 2, 6 are input nodes. This graph contains no branchpoint nodes. The other nodes are normal nodes, except node 9, which is output node.

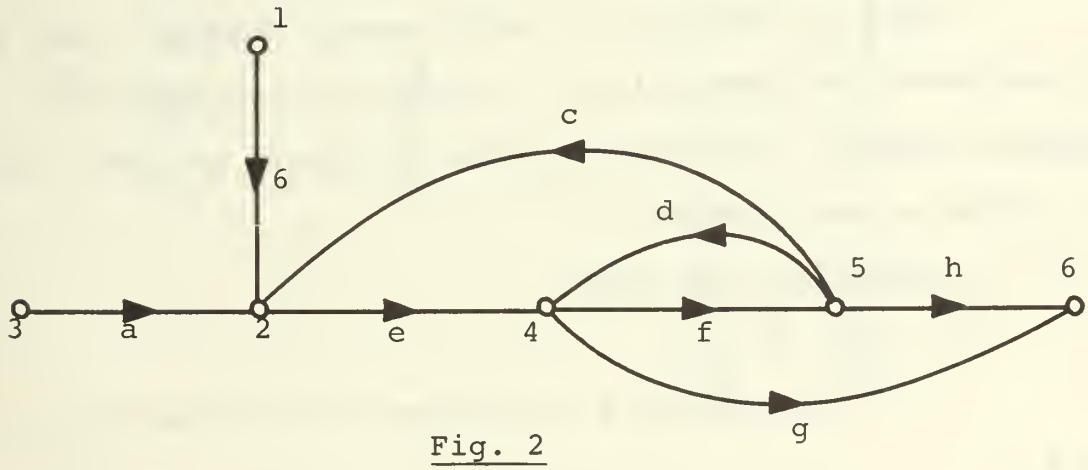


Fig. 2

In this graph node 4 is (also) a branchpoint node.

LIST1

Column 1	Column 2	Column 3
node number k	branch designator A	node number 1
from	via	to

elements in Col. 1 are referred to as LIST1 (1,k)

elements in Col. 3 are referred to as LIST1 (3,k)

LIST1 constitutes a complete listing of the graph. Columns 1 and 3 contain numbers, Column 2 contains an alpha-numeric designator in the form "letter-number", "letter" or "letter-letter".

The order of the listing is arbitrary; it is recommended to bring the list in at least approximate numerical order in Column 1 for improved error detection.

4. LIST2 and LIST3

LIST2 is identical to LIST1 except Columns 1 and 3 are exchanged and then Column 1 in LIST2 is arranged in numerical order. LIST3 has the same structure as LIST1, but the contents have changed.

5. Topological Matrices

a. Loop Matrix

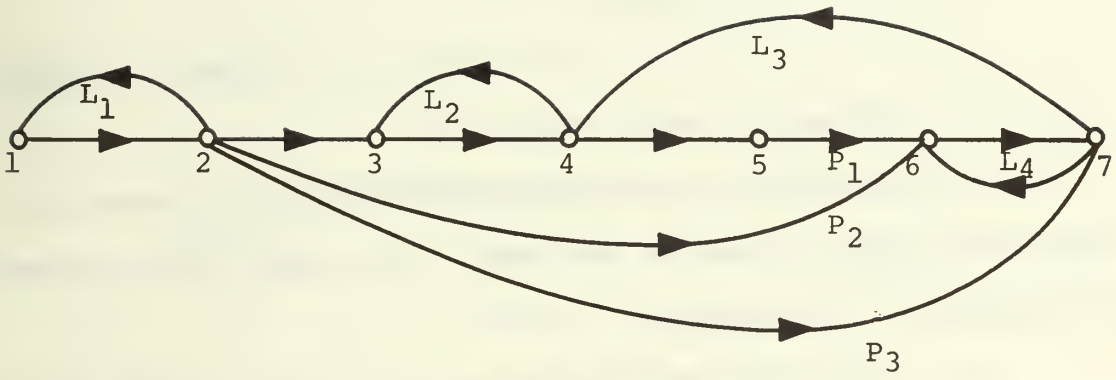
A loop matrix L is defined. The elements of L are a_{jk}.

$a_{jk} = 1$ for node k being in loop j

$= 0$ o.w.

j=1,2,3,...nl, nl = number of loops

k=1,2,3,...nn, nn = number of nodes



$$\underline{L}^T = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \qquad \underline{P}^T = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Fig. 3

b. Path Matrix P

A path matrix P is defined. The elements of P are b_{mk}.

$b_{mk} = 1$ for node k being in path m

$= 0$ o.w.

m = 1,2,3,...nn, nn = number of nodes

c. Touching Loops Matrix LT

A matrix is defined which contains information about loops which touch in the Mason flow graph. The elements of this matrix LT are d_{kj}.

$d_{kj} = 1$ for loop k touches loop j; that is loops k and j have at least one node in common.

$= 0$ o.w.

k, j = 1,2,3,...nl, nl = number of loops

d. Path - Loop Touching Matrix PL

A matrix PL is defined which indicates which paths touch which loops. The elements of PL are e_{km}.

$e_{km} = 1$ for loop k touches path m

$= 0$ o.w.

k=1,2,3,...nl, nl = number of loops

m=1,2,3,...np, np = number of paths

e. Control Vectors

Certain control vectors are defined to check on specified conditions.

(1) Loop Search Control Vector LC

The element of LC are c_{lk}.

c_{lk} = 1 for every input and output node

= 1 for every other node used at least once in the loop search

= 0 o.w.

k=1,2,3,...nn, nn = number of nodes

(2) Multiple Node Control Vector LIST1(4)

LIST1(4,k) = 0 for node LIST1(1,k) being a single node or the mth node in a series of m equal node numbers

= 1 for the first m - 1 node numbers in a sequence of m equal node numbers

k=1,2,3,...nn, nn = number of nodes

(3) Branchpoint Control Vector BC

This vector contains information about nodes which are branchpoint nodes. The information is in the form of an index k, identifying the branchpoint node in LIST2(1,k).

BC(i) = 0 for node i being not a branchpoint node

= k for node i (= LIST2(1,k) being a branchpoint node.

6. Logical Operations On Matrices

a. The Operation ⊗

By the ⊗ - product of two matrices A ⊗ B in that order of the m x p matrix A = (a_{ij}) and the p x n matrix B = (b_{ij}) is meant the m x n matrix C = (c_{ij}) where

$$c_{ij} = a_{il} \,\&\, b_{lj} \oplus a_{i2} \,\&\, b_{2j} \oplus \ldots \oplus a_{ir} \,\&\, b_{rj}$$

$$= \bigoplus_{k=1}^{r} (a_{ik} \,\&\, b_{kj}),\ i=1,2,\ldots m;\ j=1,2,\ldots n$$

where & is the logical operation AND with truth table:

a	b	a&b
0	0	0
0	1	0
1	0	0
1	1	1

and $\oplus$ is the logical operation OR with the truth table:

a	b	c⊕b
0	0	0
1	1	1
0	1	1
1	0	1

B. THE LOOP SEARCH

1. Initialization

Consider LIST1. Define two temporary control vectors T1 and T2 such that for every node there is one binary position reserved. Then set for every node k present in LIST1(1) the element T1(k) to 1 and for every node m present in LIST1(3) the element T2(m) to m.

T1(k) = 1 for LIST1(1,k) $\neq$ 0

T2(m) = 1 for LIST1(3,m) $\neq$ 0

Now the control vector LC is initially set to

LC = T1 $\bar{\&}$ T2

Example:

LIST1: 1 a 2

3 b 2

2 c 4

4 d 5

5 e 6

graph:

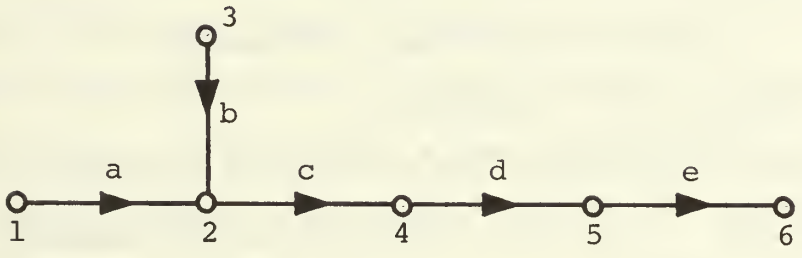


Fig. 4

T1 = [1 1 1 1 1 0]

T2 = [0 1 0 1 1 1]

LC = [1 0 1 0 0 1]

The next step in the initialization is the marking of the multiple nodes. Use is made of the control vector LIST1(4).

The control vector LIST1(4) is set to its value according to its definition. (See program FINDLOOP)

2. Search For Loops

a. Finding Successive Nodes

The Column 1 of LIST1 is called LIST1(1), the Column 2 is called LIST1(2). An element in a column of LIST1 will be denoted by LIST1 (column number, element index).

The search starts in LIST1(1), using LIST1(3,1) as the first search key. The result of this search is an index k at which:

LIST1(3,1) = LIST1(1,k)

Now the search key is changed:

search key = LIST1(3,k)

and the search continues, at the top of LIST(1) or at the next location. (For short lists it may be reasonable to start at the top again.)

When in the continuing search:

LIST1(3,k) = LIST1(1,i)

the search key is again changed to:

search key = LIST1(3,i)

and the search continues.

In a graph containing loops, LIST1 contains multiple equal node numbers in both Column 1 and 2.

The question arises how to proceed with the search when these multiple node numbers are encountered. The control vector LIST1(4) contains information about those nodes which have leaving branches in more than one direction. It is necessary to follow all possible directed paths through the

graph. LIST1(4) enables the search routine to keep track of all nodes where to try to follow also other possible paths.

If during the search an element LIST1(1,k) is encountered whose related control element LIST1(4,k) = 1, then the element LIST1(3,k) is selected as the next search key (as explained above), but the control vector element LIST1(4,k) is set to "0" before the search continues.

b. Detection Of a Loop

The element LIST1(1,1) and all the elements LIST1(1,n) which are "found" during the search are "marked" in a temporary control vector (the max. length is equal to the number of nodes). Every time a new search key is selected, a check is made against the control vector. If for the specific node a = LIST1(3,i), a "1" is found in the control vector, a loop is detected because a closed directed path had been followed, the path being closed at LIST1(3,i).

c. Recording Of the Loop

To record the actual sequence of nodes in a loop, which is an orderly way of noting the loop nodes, it is necessary to store in a list all successively "found" elements LIST1(1,n), starting with the initial value LIST1(1,1). This is done during the search and the list is abandoned if the search is terminated before a loop could be found. But most likely the actual sequence is of no consequence. After the j^{th} loop detection the "loop matrix" L is updated by filling the j^{th} row of L with the appropriate entries.

Also for each node touched during the search an entry is made in control vector LC.

d. Completion Of the Loop Search

The loop search is basically terminated if by the process described in b. above no loop is detected. However, before it can be assumed that all loops of the graph have been found, the control vector LC must be checked. If LC = 1 (contains only "1" 's), all nodes in the graph have been checked for belonging to a loop. Otherwise the search must be continued. For practical purposes the search could start at any node which was not yet included in the search and which is identified by a "0" in LC.

The newly found loops are recorded only if they are different from the ones already found. This condition can be detected easily, using logical functions of the programming language, by comparing the new j^{th} entry row for matrix L with the other rows of L already set.

The multiple starting of a search is necessary if the search was started in a part of the graph which is connected to the rest of the graph only by branches coming from the other part, as illustrated below in Fig. 5.

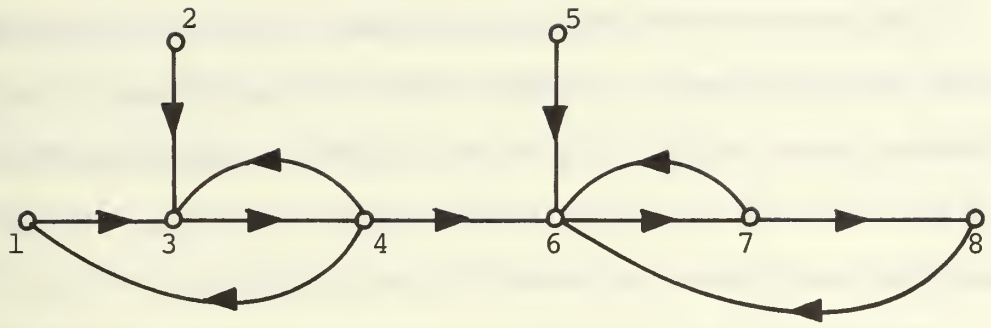


Fig. 5

In the graph given (Fig. 5) a start at nodes 5, 6, 7, or 8 could not lead to the detection of the loops 1-3-4 and 3-4. On the other hand a start at node 2,1,3, or 4 would lead to the detection of all loops in one search.

Here, how fast the search will be completed depends on the organization of the input information by the user.

C. THE PATH SEARCH

1. Initialization

For the purpose of finding all paths of the graph leading from a specified node to another specified node, a reorganized input list is formed. This list is called LIST2.

As a first step a multiple node control vector LIST2(4) is formed, exactly analogous to LIST1(4).

2. Path Search

The path search follows the same logical sequence of events as the loop search, except that the path search starts not at the element at the top of the list but with LIST2(1,a), using LIST2(3,a) as the first search key. The node "a" is the specified end node, to which a path is sought.

The result of the search using the search key is an index k at which

LIST2(3,a) = LIST2(1,k) .

Now the search key is changed:

search key = LIST2(3,k)

and the search continues as described before (B.2). This is the basic description of the path search. The difference , compared to the loop search, is the fact that here the predecessors of the end node are found and that here all connections are avoided which would result in loops or which would not end at the desired starting node.

a. Detection Of a Path

A path is detected when it is found that the search key is set to an element LIST2(3,i), where the node "i" is the specified starting node. The path so detected leads from node "a" to node "i", or, in the notation used here to facilitate computer programming, from LIST2(1,a) to LIST2(3,i).

During the search a temporary control vector is updated with "1" entries as described above (2.b) for the loop detection. Here the element LIST2(1,a) and all the

elements LIST2(3,n), which are used as search keys are marked with a "1" in the related place in the control vector. If for the element LIST2(3,i) a "1" is already marked in the control vector, a loop is detected. This is an unwanted situation. The loop must be left by going back to the last multiple node. Then one of the other alternate paths going back from that node must be followed.

How is it possible to keep track of the multiple nodes passed? It must be remembered, that the multiple nodes are marked in LIST2(4). (See definition for the analogous vector LIST1(4)). So every time a new search key is selected, LIST2(4) will be checked, and a "1" in the position k, which is the "found" - index, indicates that the node LIST2(1,k) is a multiple node. When passing these nodes during the search, the element LIST2(4,k) is set to "0" so that it cannot be found again.

The control vector BC (for Branch Control) is used to store all multiple nodes passed during the search, probably best in the form of the index, where the multiple node can be found in LIST2(1). It is then possible to identify the node to which one must go back to resume the search, in case a loop is detected, or the path detected ends at a node not specified as the end node.

The latter case is possible and is identified by the fact, that using the present search key, (the last one assigned) no element LIST2(1,i) can be found in column LIST2(1) which is equal to the search key.

b. Recording Of the Path

To record the actual sequence of nodes in the path, backwards from the end to the start, it is necessary to store in a temporary list all the elements LIST2(3,n) used as search keys during the search and the initial element LIST2(1, a). This list contains only those nodes which actually form the path. Intermediate restarts of the search are not recorded.

After the search for the j^{th} path is completed, the j^{th} row of the matrix P is filled with the appropriate entries. This can be most practically accomplished by defining the temporary list mentioned above in the form of a row of P. Then it is only necessary to set the j^{th} row of P equal to this temporary list at the end of the search.

D. EVALUATION OF TRANSFER FUNCTIONS

To evaluate the transmission gain from an input node "i" to an output node "o" (here: a node, which must have at least one incoming branch) the Mason Gain formula is used:

$$G = \frac{\Sigma \Delta_k \, P_k}{\Delta}$$

where P_k is the gain of the k^{th} direct path from node "i" to node "o"

Δ_k is the cofactor of the k^{th} path, formed from Δ by striking out all terms containing loops which are touched by the k^{th} path.

1. Evaluation Of the Graph Determinant Δ (DELTA)

If the graph would contain only loops which do not touch, then the graph determinant would be evaluated as

$$DELTA = (1-L_1)(1-L_2)\ldots(1-L_i)$$

where i = nl, the number of loops in the graph, and L_j is the loop gain, the product of the gains of the branches which form the loop L_j.

If touching loops are present, then the products of their loop gains are eliminated from the expression for DELTA.

a. Evaluation Of the Matrix LT

The matrix LT contains information about which loops touch each other.

The matrix LT is evaluated as:

$LT = L \otimes L^T$ L^T is the transposed matrix L, ⊗ is the operation defined in I.6.a

Proof. Two loops touch each other when they have at least one node in common. By definition of the matrix L and the operation ⊗ an element LT(k,j) = 1 if at least one of the logical expressions a_{ik} & b_{kj} = 1. There are k of these expressions, for every node in the graph.

By definition of the operation " & " an expression a_{ik} & b_{kj} = 1 only if a_{ik} = 1 and also b_{kj} = 1. But this is the case only if the node k is in loop i (for a_{ik}) and also in loop j (for b_{kj}). Consequently LT(i,j) = 1 only if one of the graph is in loop i and loop j simultaneously. This completes the proof.

It should be noted that the matrix LT is symmetric and has always all elements on the main diagonal equal to 1. As a consequence of this all the elements above or below the main diagonal and the elements on the main diagonal could be discarded. The computer program, Mason which is used to evaluate DELTA (Program 2) uses this fact.

For convenience the example of Fig. 3 is given:

$$L = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

then LT is evaluated:

$$LT = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

Now LT gives the result:

L_2 and L_3 touch,

L_3 and L_4 touch.

The number of touching loops in the graph (of order 2) is equal to the sum of the elements of LT divided by two.

b. Evaluation of DELTA By a FORMAC Program

A computer program, using the language FORMAC, is used to evaluate the expression for DELTA. (Program 2)

The program is only meant as a demonstration of how this problem of evaluating DELTA can be solved using an algebraic manipulation language.

After each multiplication by $(1-L_i)$ the expanded result is checked for the presence of products of two touching loops, and if such a product is found, it is replaced by zero. It is not necessary to check for products of more than two touching loops, because larger products containing the smaller one will go to zero automatically, after the smaller product is set to zero.

To find the sets of two touching loops, the upper triangular part of LT, excluding also the main diagonal, is read row by row. The appearance of a "1" during that process signals a touching of the respective loops, indicated by the row- and column index of the "1" found. This information is stored in a table in the form of an algebraic expression consisting of the product of the two loops involved. From this table the expressions are taken when, after the multiplications mentioned above, a check for their presence in the accumulated product is made.

2. Evaluation Of the k^{th} Cofactor Δ_k (DELTK)

Having evaluated DELTA, the graph determinant, the k^{th} cofactor is evaluated in a similar way, using the same

computer program. Instead of the matrix LT now the path-loop touching matrix PL is used as input.

The matrix PL is evaluated as:

$$PL = L \otimes P^T$$

where P^T is the transposed path matrix P. The resultant matrix PL shows the path - loop connections. If the matrix is read row-wise, then for each row i and column j it is indicated by a "1", if loop i touches path j. The proof follows the proof for the case of the touching loop matrix LT.

For the example given for the loop case: (Fig. 3)

$$PL = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

which indicates, that only loop 2 is not touched by path 2 or path 3.

Now two possible ways of solution can be followed. One way is to analyze the matrix PL in a way similar to the one used for the matrix LT before. For each column of PL a specific cofactor $DELTK_k$ is evaluated. For a "1" in row i, loop L(i) is eliminated from the previously evaluated expression for DELTA. Now the simple replacement of L(i) by zero results in elimination of all terms in DELTA which contain that loop L(i). After processing in that way all columns of PL, all cofactors are evaluated. Now DELTA and the evaluated cofactors $DELTK_k$ for the k paths present can be used in the final evaluation.

The other possible solution is first to use DELTA in the final evaluation by substituting for every L_i in DELTA the actual algebraic expression for L_i and then evaluate the cofactors new from the original expression involving the accumulated products of $(1-L_i)$, similar to the evaluation of DELTA.

3. Evaluation Of the k^{th} Path Gain (PK)

The algebraic evaluation of the path gain is best done immediately after a path is found (see II.C.2.b). One method of solution would be to store during the search for the path also the index k of the different search keys. This index sequence could then be used in the algebraic part of the program to find those branch gains in the input table LIST1 which must be multiplied to form the path gain.

Two basic approaches are possible:

- the symbols stored in LIST1(2) represent larger and more complex algebraic expressions. Then they constitute names or, to use FORMAC vocabulary, they are FORMAC variables. In this case there must be an assignment of the algebraic expression to that variable somewhere in the program. (See example below)
- the symbols stored in LIST1(2) are the actual algebraic expressions which constitute the branch gains. In FORMAC vocabulary they are called FORMAC expressions. In this case there

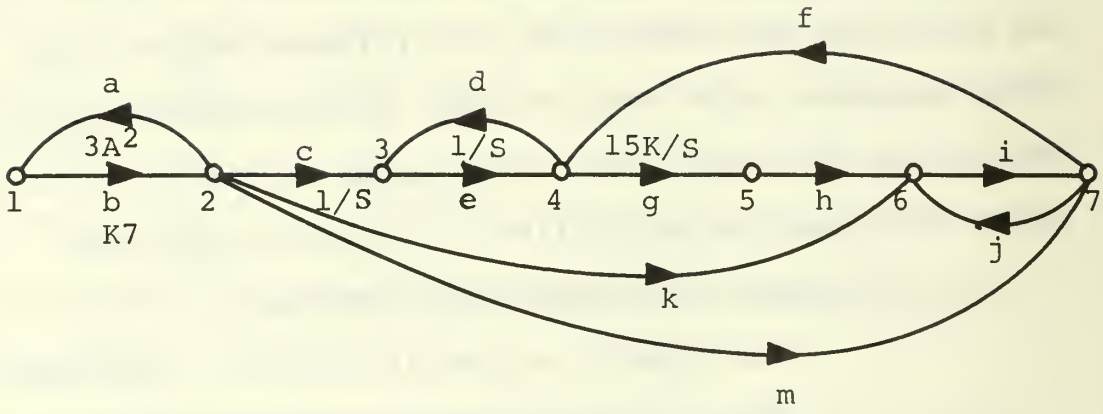


Fig. 6

must be an evaluation which calculated the path gain directly. (See example below)

Example.

Case 1.

LISTl(1)	LISTl(2)	LISTl(3)
1	b	2
2	a	1
2	c	3
2	k	6
2	l	7
3	e	4
4	d	3
4	g	5
5	h	6
6	i	7
7	f	4
7	j	6

Note: the elements in LISTl(3) must be character type.

In PL/l this is indicated by writing for the character b the expression 'b'. (See PL/l reference manual)

A FORMAC statement like the following would produce the path gain in terms of the letters in LISTl(3):

```
LET(PK(l) = "LISTl(2,l)" * "LISTl(2,3)"
                * .... * LISTl(2,i) );
```

The following statement would produce this output:

```
PRINT_Out(PK(l));
```

PK(l) = B C E ... J

Because generally it would not be known explicitly which symbols out of LISTl(2) form the gain PK, the following assign procedure would be used:

At the time a more specific evaluation in terms of the actual algebraic expressions has to be produced, an input

statement is executed. This statement would cause the read-in of the actual algebraic expressions. Then a replacement would take place, in which the letters a,b,... are replaced in the gain expression PK(1) = B C E G ... F by the algebraic expressions. Then a further expansion of the gain expression could take place.

On data cards:

'K7' '3 * A * * 2' '1/S' .....

Then the statements:

```
 DECLARE EXPR CHARACTER(80) VARYING;
                    to define EXPR a character variable.
                    to define an index.

 K = 0;
A:  GET LIST(EXPR); K = K+1;
 ON ENDFILE(SYSIN) GO TO END;
 LET(PK(1) = REPLACE(PK(1), "LIST1(2,K)", "EXPR") );
 GO TO A;
END: LET(PK(1) = EXPAND (PK(1)) );

 PRINT_OUT(PK(1));
```

The printout would read:

PK(1) = K7 3 A^2 / S

In this case the further expansion has no effect, because there are no expansions to be performed.

Now consider case 2:

LIST1(1)	LIST1(2)	LIST1(3)
as before	K7 3*A**2	as before
.	1/S	.
.	.	.
.	.	.

In this case there are no data cards. The statements:

Note: NB = number o- branches in a path

```
LET(PK(1) = 1);
DO K = 1 TO NB;
LET(PK(1) = PK(1) * "LIST1(2,INX(K))" );
END;
PRINT_OUT(PK(1));
```

Note: INX is an array of indices which indicate the branch to be included in the product.

In the example given the array INX would contain for the first path PK(1) the following indices:

```
INX(1) = 1
INX(2) = 3
INX(3) = 6
INX(4) = 8
INX(5) = 9
INX(6) = 10
```

and NB = 6.

A similar evaluation has to take place for the loop gains, the values of DELTA and DELTK.

4. Final Evaluation Of the Transmission Gain

The transmission gain G can finally be evaluated using a FORMAC program. Depending on the complexity of the problem first G could be evaluated in terms of the branch gains, where the branch gains are represented by their variable names, and by the loop gains. In this case G would be an algebraic

expression such as

$$G = \frac{(L(i)L(k)...) * (A,B,C...)}{(L(a)L(b)...)}$$

That is, DELTA and DELTK(i) would be in terms of the loop gains; L(i) and PK(i) would be in terms of the branch gains a,b,c...

Then after G is evaluated as above, a substitution could be made, where, again depending on the complexity involved in possible applications, several methods are possible. In this process the FORMAC functions REPLACE, CHAIN, EVAL and EXPAND[1] would probably have to be used. Generally a compromise has to be made between using more time for execution and less core space and vice versa.

III. DIRECT EVALUATION OF DETERMINANTS USING FORMAC

In this chapter the direct evaluation of determinants using FORMAC programs is shown. By "direct" evaluation is meant a process of evaluation where the properties of determinants are used rather than graph theory as used in chapter II for the evaluation of DELTA (which is a determinant).

A. A FORMAC PROGRAM USING THE BASIC DEFINITION OF THE DETERMINANT

In numerical mathematics a determinant is normally evaluated by normalizing the rows or columns and using elementary row- or column operations to eliminate all elements off the main diagonal. Then the value of the determinant is the product of the elements on the main diagonal. If this process would be used in the evaluation of determinants with algebraic terms as elements, the generation of rational functions or otherwise more complex algebraic expressions would create additional complexity. The handling of rational polynomial expressions, when the polynomials are in symbolic form, is very time consuming.

The program DTERM uses the basic definition of the determinant:

If A is a square matrix of order n, then the determinant of A is DET(A):

$$DET(A) = \sum_{j} (-1)^{t} a_{1j_1} a_{2j_2} \ldots a_{nj_n}$$

where the second subscripts j assume all possible arrangements in which each column is represented exactly once in each term

of the sum, and the exponent, t, is the number of interchanges necessary to bring the second subscripts into natural order (that is, 1,2,3, ..., n).

In other words, all possible permutations of the numbers 1,2,3, ..., n form the complete set of second subscripts mentioned above.

If the number t is even, the permutation is called "even", otherwise "odd".

The permutation of a set of numbers soon exceeds the time available for execution of any program. However, for n = 2 to n = 9 one may find it possible to use this method. The algorithm used in DTERM to find all the permutations of the numbers 1,2,3, ..., n, for different n has the following execution time:

n = 4 : 0.05 sec

n = 5 : 0.09 sec

n = 6 : 0.31 sec

n = 7 : 1.95 sec

n = 8 : 15.12 sec

n = 9 : 137.95 sec

In the example program DTERM the permutation of column indices is only used to demonstrate the algebraic part of the program. Normally this algebraic part would be used with another method for finding the elements of the matrix A which have to be multiplied to form a term in the expansion of the determinant. For very large matrices, (for example, a 37 x 37 matrix) the graph analysis as described before would

identify those elements. Also the method used by Lipson [4] could possibly be used to identify those elements. How much computational effort each method requires in terms of time and space has to be determined.

1. General Description Of the Evaluation Of the Determinant By the FORMAC Program DETERM

a. The Non-Algebraic Part Of DETERM

The first step after the initialization is the evaluation of a permutation of the numbers 1,2 ..., n. The program used for that purpose is the subroutine PERM[1]. The program PERM produces upon each call another distinct permutation of the n ! possible permutations in such a way that the permutations it generates are alternately odd and even, a property the author of algorithm 115 PERM described as "probably useless". However, this property enables us to determine the sign of the permutation.

Having found a specific permutation it is used as the set of indices j1, j2, ..., jn mentioned in the definition of the determinant. At the same time the sign of the product $a_{1_{j_1}} a_{2_{j_2}} \ldots a_{n_{j_n}}$ is known.

The elements a_{i_j} in the product are then tested for being zero. If a zero element is found, a new permutation is produced and the process starts again, because further algebraic manipulation would only produce a zero term.

---

[1]The authors conversion of the algorithm 115 PERM, H. F. Trotter, Comm. ACM, Aug. 1962 from ALGOL into PL1.

In case the elements are all non-zero the product is algebraically evaluated. This is a complicated step, which will be described in the next section.

Then the next non-zero term (a term is one of the products $a_{iji}a_{kjk}\cdots a_{njn}$) is evaluated in the same manner.

Finally all coefficients of equal powers of the polynomial variable in the different terms of the determinant are collected.

b. The Algebraic Part of DETERM

Most of the algebraic manipulation statements are detectable by the word "LET(.....); and the embracing brackets. The semicolon is a feature of PL/1; it signals the end of a statement.

Consider now two polynomials in the variable S:

$$P1 = (A + B^2)^2 S + (C + D(E + F)^3 S^2$$

$$P2 = (A + C)^3 S + B S^3 + F(G-H) S^5$$

If the product P1*P2 is formed and all the terms are completely expanded, a new polynomial is formed which has 90 terms before simplification.

If the same product is formed without expanding the coefficients of the variable S the resultant polynomial has only 6 terms. Each of the coefficients of the variable S is in unexpanded form.

Example.

$$P1 = (a + b) s + (c + d) s^2$$

$$P2 = (e + f) s + (g + h)$$

The expansion used in DTERM:

$$P1\ P2 = (a + b)(e + f)\ s^2 + (a + b)(g + h)\ s + (c + d)(e + f)\ s^3 + (c + d)(g + h)\ s^2$$

Then this result is simplified to

$$P1\ P2 = (a + b)(g + h)\ s + ((a + b)(e + f) + (c + d)(g + h))\ s^2 + (c + d)(e + f)\ s^3$$

or $$= A\ s + B\ s^2 + C\ s^3$$

where the coefficients A, B, C ... are stored in this form on disks. After all terms of the determinant are expanded in this manner, coefficients of equal powers of s are read back into main core and are expanded fully and added. They are then printed and the space used by them is subsequently made available to other uses.

The latter method is used in the first part of the algebraic manipulation to save space. A main limitation of FORMAC (probably of any algebraic manipulation language) is the enormous space requirement for certain expansions; therefore to make algebraic manipulation of large expressions possible at all, it must be done at the expense of time. It. has been shown [6] that running FORMAC programs can be made much faster after some refinement; often the omission of the use of one FORMAC function showed a considerable increase in execution time. The reason is that each FORMAC function use constitutes really the use of a subroutine.

The main emphasis in DETERM was placed on the possibility of evaluating very large determinants, the refinement was secondary.

After all n factors, where each is considered being a polynomial, have been multiplied, the resultant product, which is called "PD" in the program, is further processed.

Each coefficient in PD of the variable S and the constant coefficient is identified and stored separately on disk file. Then the evaluation of a new term PD starts.

After all terms PD, which are the products mentioned in the definition of the matrix, have been processed in the manner described above, the coefficients stored on disk are now retrieved and added up, always adding coefficients of the same power of the variable S originating from the different terms of the determinant. The coefficients are designated as follows:

$$COEF(i) = C(1,i) + C(2,i) + \ldots + C(k,i)$$

where i is the power of S which is under consideration and the numbers 1,2, ...k are the numbers of the k non-zero terms in the determinant.

The coefficients C_{ki} are the ones which are stored on disk. The final coefficient $COEF_i$ is the coefficient of the variable power S^i in the expanded determinant.

Example for using DTERM[2]

System equations:

$$e_i = k_{pi}\ \theta_i$$

$$e_o = k_{po}\ \theta_o$$

$$e = e_i - e_o$$

$$e_f = k_a\ e_a$$

$$e_t = h\ \dot{\theta}_t$$

$$e_a = e_c - e_t$$

$$\lambda_m = k_t\ i_f\ i_a$$

$$\frac{E_c(s)}{E(s)} = G_c = \frac{s + a}{s + b}$$

The state equations are formed with the following variables as state variables:

$$X_1 = \theta_o$$

$$X_2 = \omega_o$$

$$X_3 = i_f$$

$$X_4 = \text{dummy variable}$$

$$G_c(s) = \frac{E_c(s)}{E(s)} = \frac{X_4}{E}\ \frac{E_c}{X_4} = \frac{1}{s + b} \cdot \frac{s + a}{}$$

---

[2]Example taken from notes course 3411 Naval Postgraduate School.

which gives:

$$s\ X_4 = -\ (b)\ X_4 + e$$

$$e_c = \dot{X}_4 + a\ X_4$$

The equations in the S - domain:

$$(1)\ s\ X_1 - X_2 = 0$$

$$(2)\ (jo(\rho_1\ \rho_2)^2 + Jm)\ s\ X_2 + (fm(\rho_1\ \rho_2)^2 + fm)\ X_2 - (\rho_1\ \rho_2\ k_t\ i_a)\ X_3 = 0$$

$$(3)\ (k_a\ k_{po})\ X_1 + (k_a\ h\ l/\rho_2)\ X_2 + (r_f)\ X_3 + (L\ s)\ X_3 + (k_a(b - a))\ X_4 = k_a k_{pi} \theta_i$$

$$(4)\ (s)\ X_4 + (k_{po})\ X_1 + (a + b)\ X_4 = k_{pi} \theta_i$$

The equations in matrix form:

$$\underline{A}\ \underline{X} = \underline{B}$$

where A is a matrix of polynomial coefficients

X is a matrix of variables

B is a matrix of constants

The elements of A as used as input to DTERM:

$$A_{11} = s$$

$$A_{12} = -\ 1$$

$$A_{22} = (jo(\rho_1\ \rho_2)^2 + Jm)\ s + (fm(\rho_1\ \rho_2)^2 + fm)$$

$$A_{23} = -\ (\rho_1\ \rho_2\ k_t\ i_a)$$

$$A_{31} = (k_a \; k_{po})$$

$$A_{32} = (k_a \; h \; 1/\rho_2)$$

$$A_{33} = (L \; s + r_f)$$

$$A_{34} = (k_{po})$$

$$A_{44} = (s + a + b)$$

The program DTERM with the matrix A as input and the output is contained in section COMPUTER PROGRAMS.

Figure 8 shows the physical system, Fig. 9 shows the Mason flow graph.

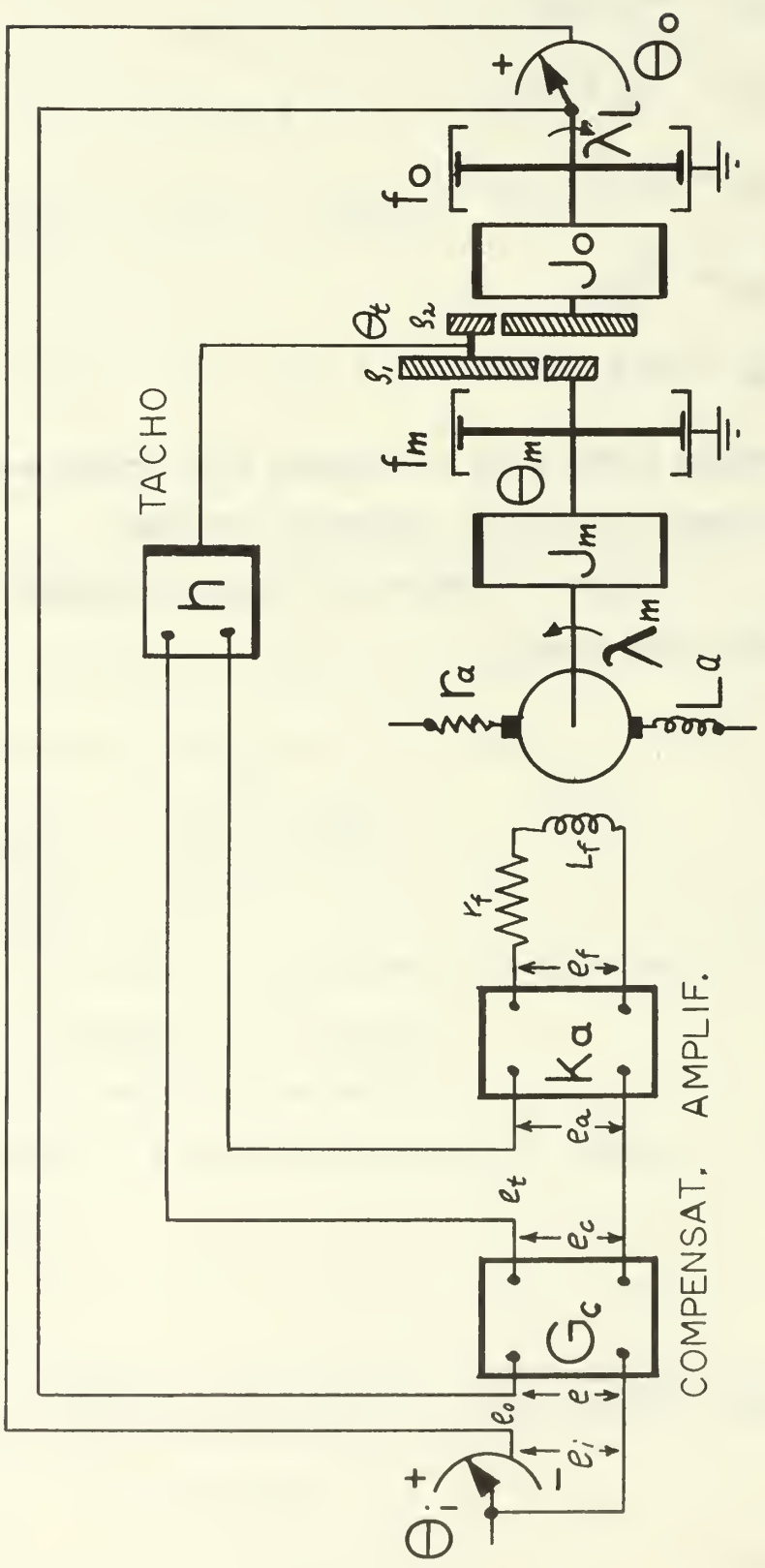

COMPENSAT. AMPLIF.
TACHO
h
Gc
Ka
Jm
Jo
fm
fo
ra
La
Lf
rf
ei
eo
e
et
ec
ea
ef
θi
θo
θm
θt
s1
s2

Figure 8

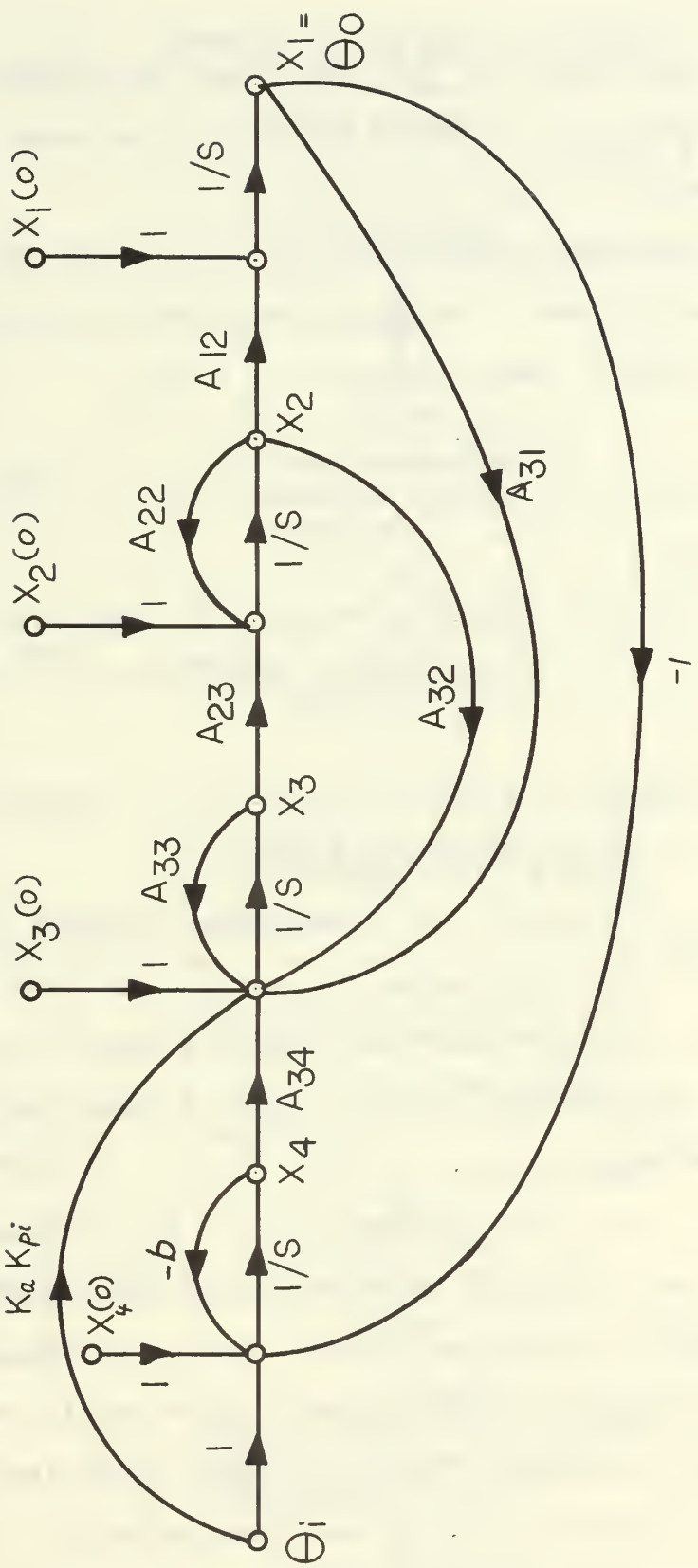

$X_1 = \Theta_0$
$X_1(0)$
1
1/S
A12
X_2
$X_2(0)$
1
A_{22}
1/S
A_{31}
A_{23}
A_{32}
X_3
$X_3(0)$
1
A_{33}
1/S
A_{34}
X_4
$K_a K_{pi}$
$X_4(0)$
1
-b
1/S
1
Θ_i
-1

Figure 9

2. Discussion Of the Flow Graph For DETERM

The flow graph for DETERM is discussed here, to some limited extent.

For clarification the definition of the determinant is now again given in terms of the symbols used in the program.

$$DET(A) = PD_1 + PD_2 + \ldots + PD_{in}$$

where

$$PD_j = C(j,i)\ S^i \qquad i = 0,1,2$$

and finally

$$DET(A) = COEF(0) + COEF(1)\ S + COEF(2)\ S^2 + \ldots + COEF(MXPOW)\ S^{MXPOW}$$

where

$$COEF(i) = C(1,i) + C(2,i) + \ldots + C(IN,i)$$

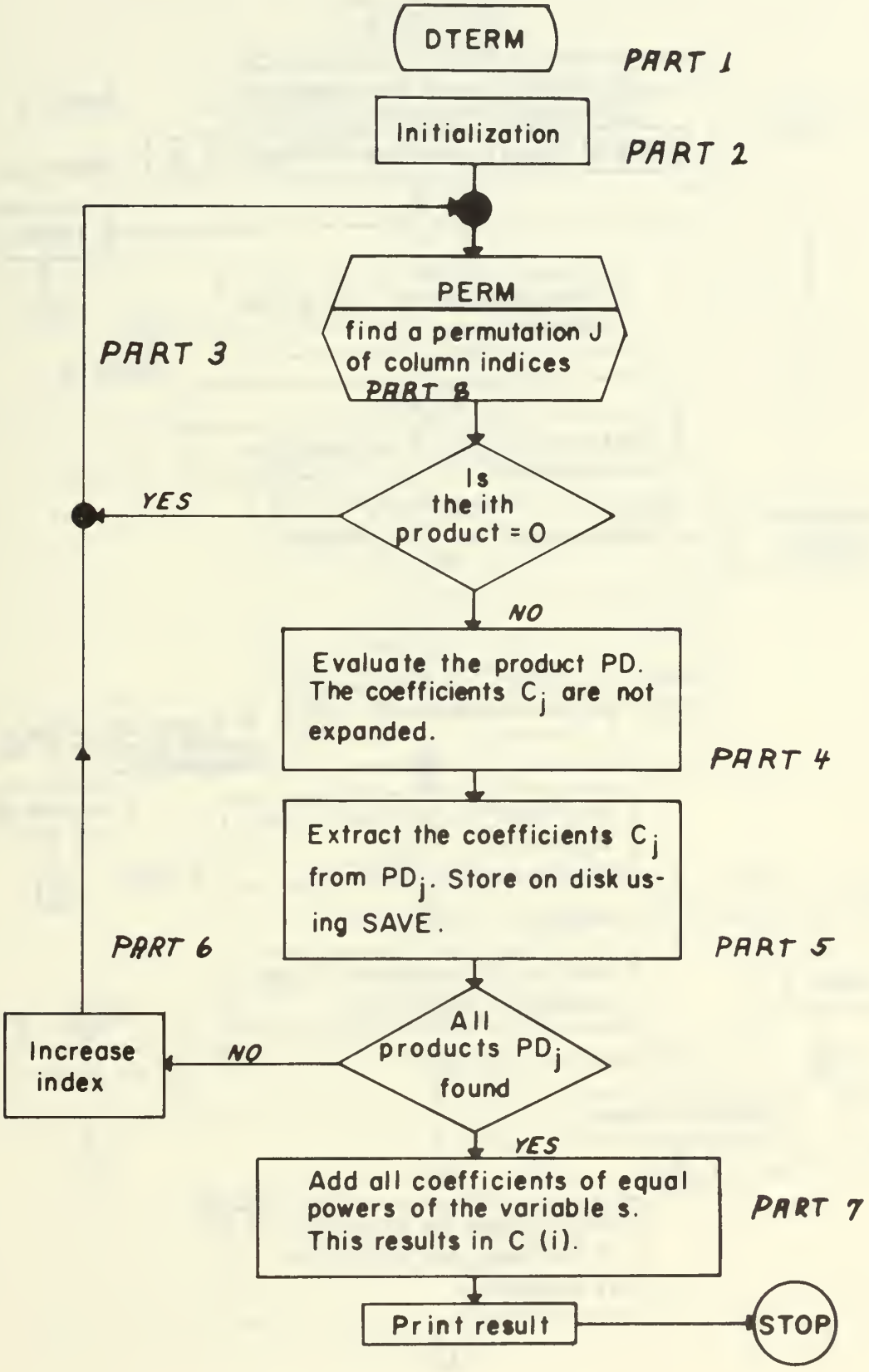
a. Data Design For DTERM

Consider now the computer program printout. (Program 1). All statements beginning with "DCL" are declarations, they define those data elements used in the program which are defined explicitly. Some of these definitions will now be discussed.

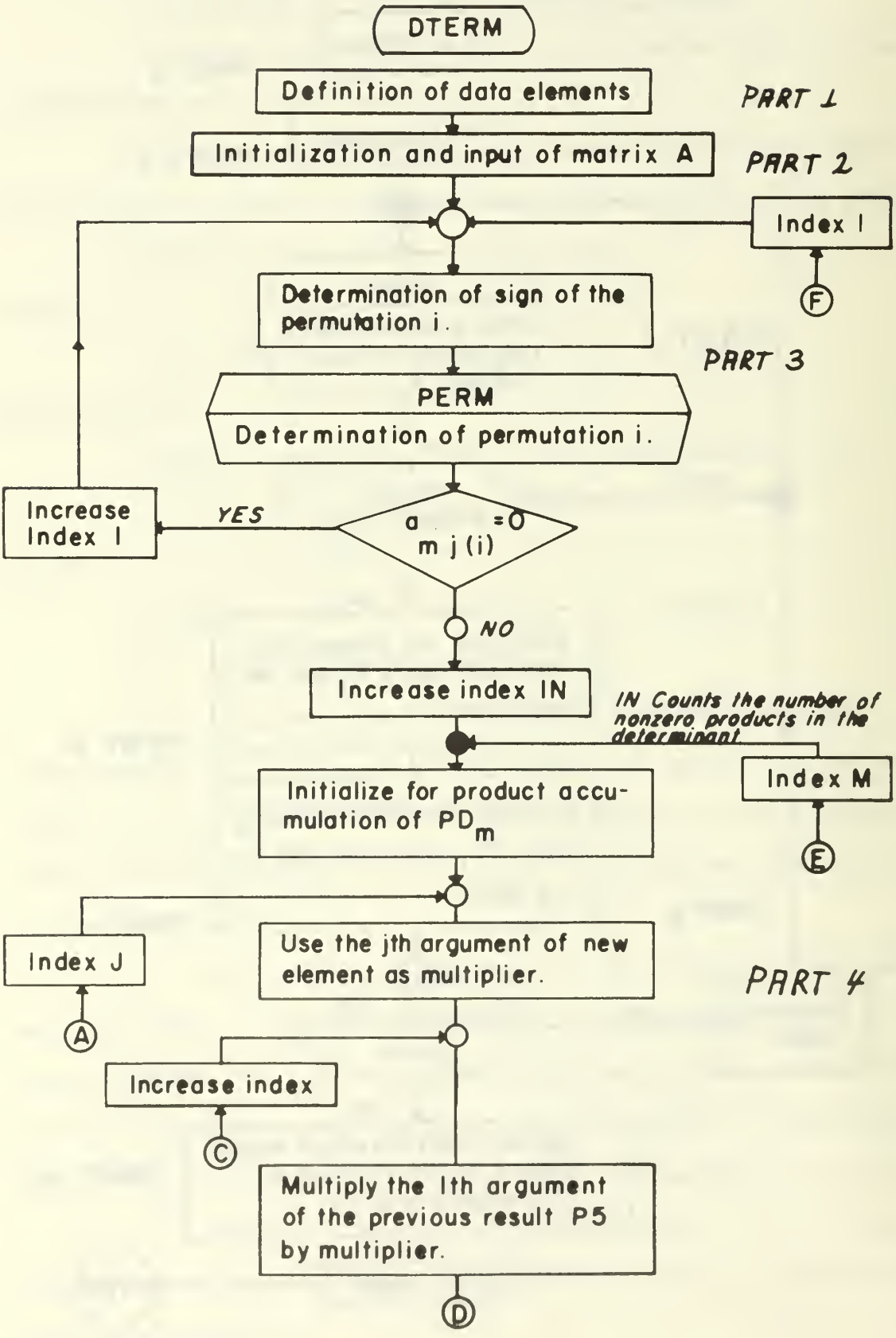
IP is an integer array. It contains after each call of PERM a distinct permutation of the numbers 1,2,...n. These numbers are used as column indices in designating elements of the input matrix A. The array A is a matrix of characters. Each element can have a varying length of maximal 75 characters.

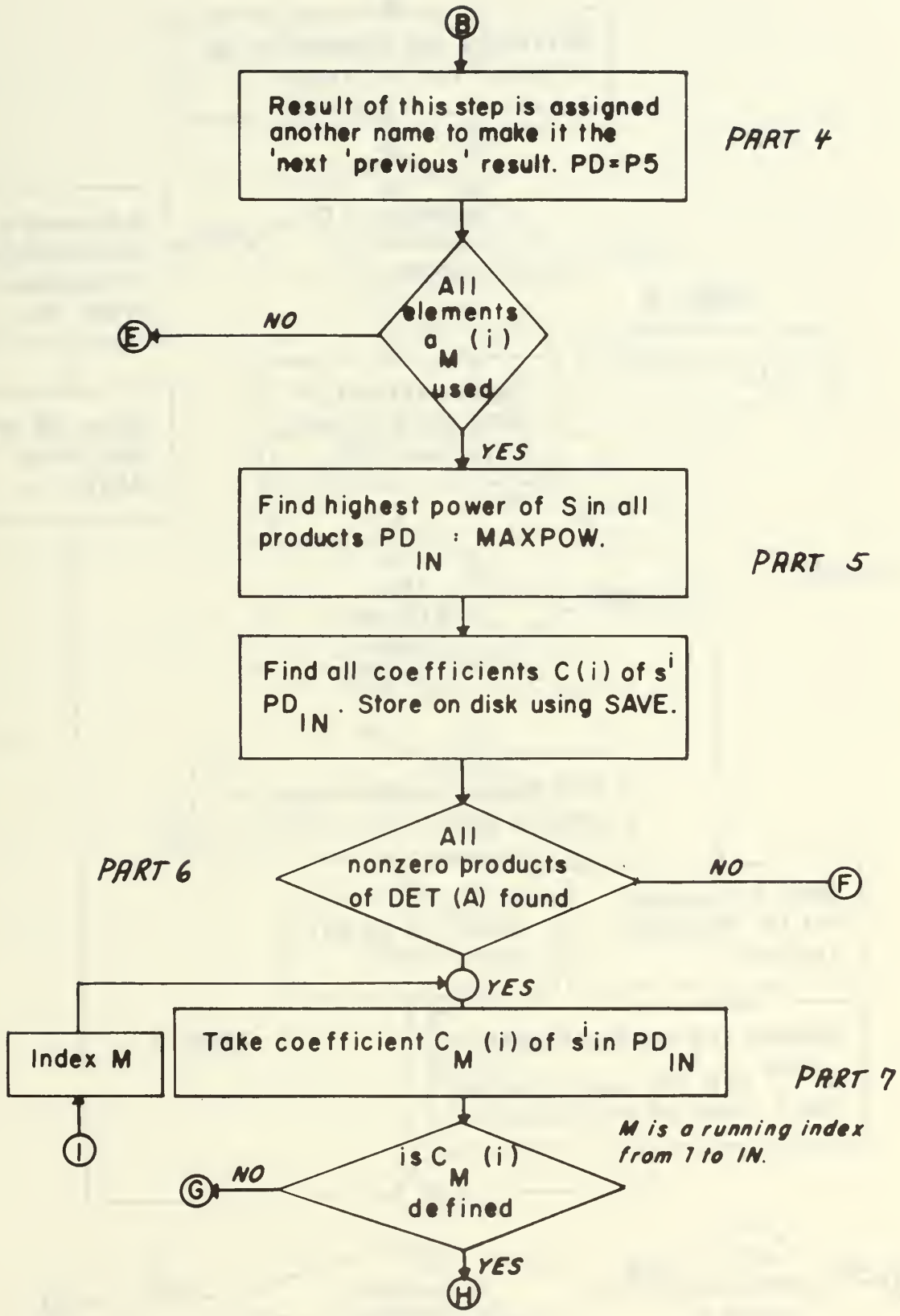
PDD is a character type variable. It contains at certain moments an element of the matrix A.N1 is the order of the input matrix A.

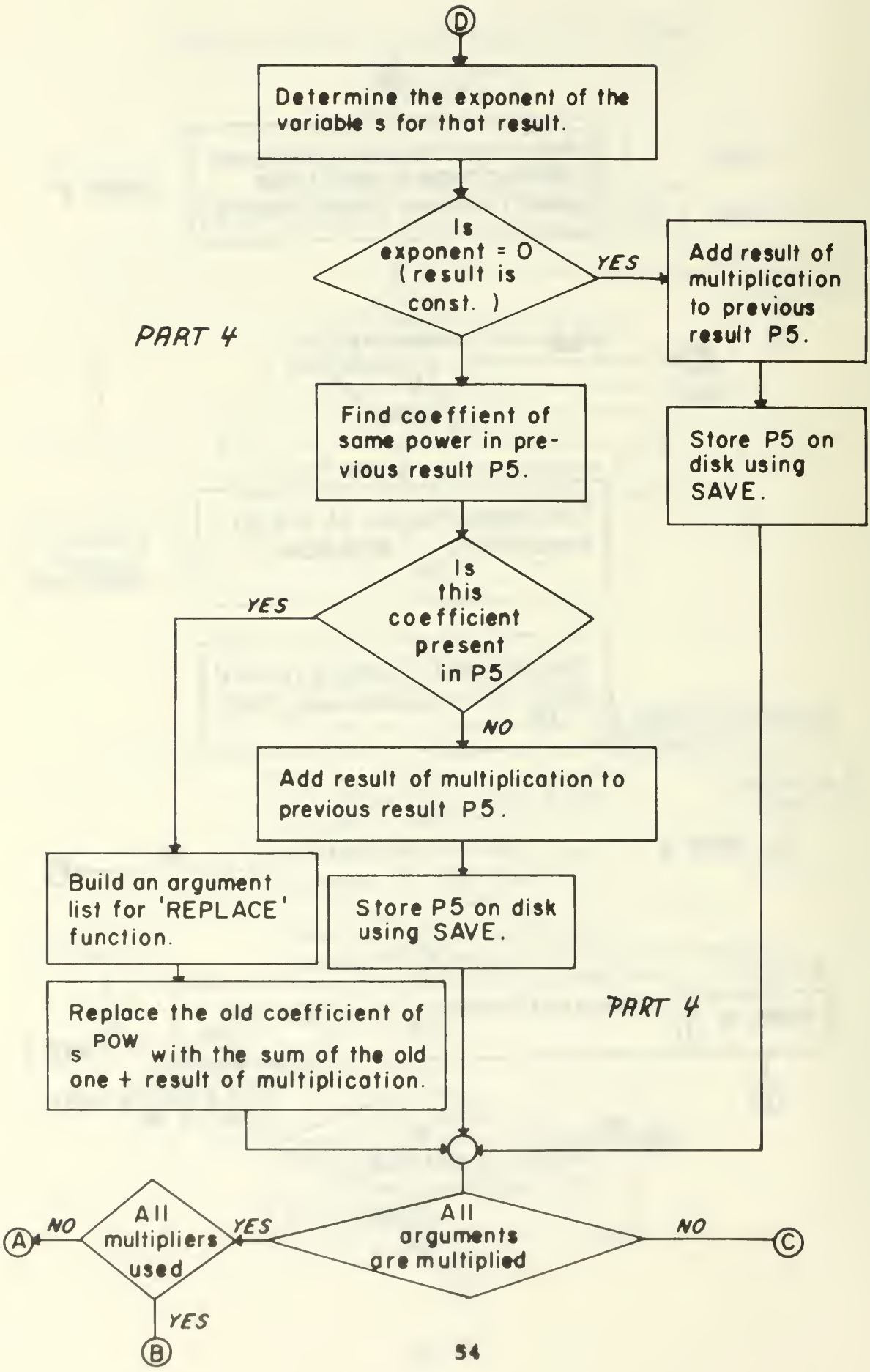
General flowgraph of DTERM

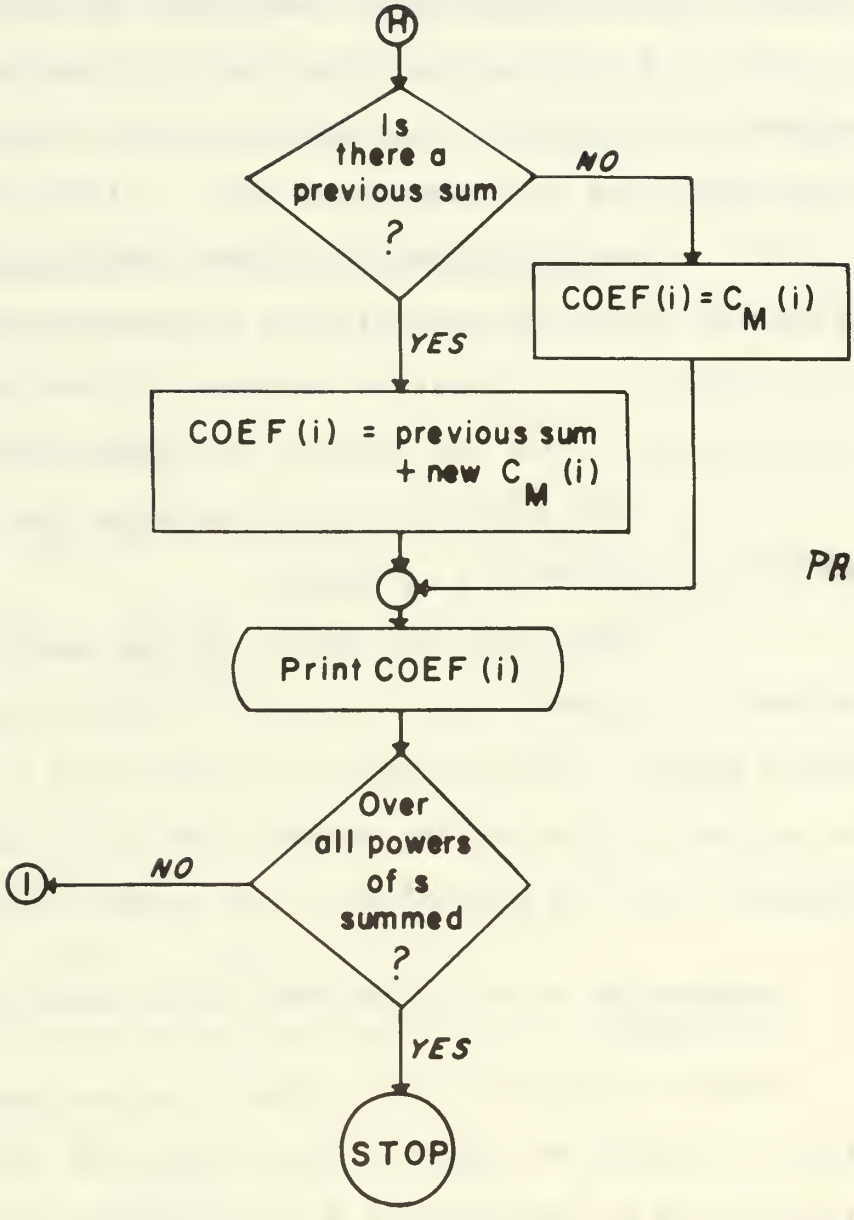


SPECIALIZED FLOWCHART OF DTERM




B
Result of this step is assigned another name to make it the 'next 'previous' result. PD=P5
PART 4
All elements $a_M(i)$ used
NO
E
YES
Find highest power of S in all products PD_{IN} : MAXPOW.
PART 5
Find all coefficients C(i) of s^i PD_{IN}. Store on disk using SAVE.
PART 6
All nonzero products of DET (A) found
NO
F
YES
Index M
Take coefficient $C_M(i)$ of s^i in PD_{IN}
PART 7
M is a running index from 1 to IN.
I
is $C_M(i)$ defined
NO
G
YES
H


D
Determine the exponent of the variable s for that result.
Is exponent = 0 (result is const.)
YES
Add result of multiplication to previous result P5.
PART 4
Find coeffient of same power in previous result P5.
Store P5 on disk using SAVE.
Is this coefficient present in P5
YES
NO
Add result of multiplication to previous result P5.
Build an argument list for 'REPLACE' function.
Store P5 on disk using SAVE.
PART 4
Replace the old coefficient of s POW with the sum of the old one + result of multiplication.
All multipliers used
NO
YES
All arguments are multiplied
YES
NO
A
B
C


H
Is there a previous sum ?
NO
YES
COEF (i) = C_M (i)
COEF (i) = previous sum + new C_M (i)
Print COEF (i)
Over all powers of s summed ?
NO
1
YES
STOP

PART 7

b. The Logic Sequence (See flow chart, Fig. 9)

The initialization includes some statements which are self explanatory. Some will be discussed now. N2 is used in a loop to fill the array IP initially with the sequence of numbers 1,2, ,,,n in natural order, an input requirement for the subroutine PERM. Also the binary variable "FIRST" is used for control of PERM. The following loop, using the index I=1 to N! is the main loop of the program. In that loop all n ! possible products of the determinant are investigated. Then the sign of the permutation is determined.

The loop M=1 to N=1 includes the zero check for the i^{th} product of n ! products.

The next loop M=1 to N1 is reached only if all factors in a product are non-zero. Then the algebraic manipulation begins. The procedure was described in 2. before The action of the various FORMAC functions (NARGS, ARG, LOWPOW, ATOMIZE, etc.) is described in the FORMAC reference manual [6].

B. EVALUATION OF A DETERMINANT USING BARREISS FRACTION-FREE ALGORITHM

Another method of evaluating a determinant is using an algorithm given by Lipson. This algorithm is based on a two-step variant by Barreiss of a fraction-free elimination algorithm, which in turn is a modification, attributed to Jordan, of Gaussian elimination [4].

Lipson uses the algorithm to evaluate a system of linear algebraic equations by transforming a matrix into triangular

form and following back-substitution. The whole process is fraction-free.

The linear equations processed have symbolic coefficients in the form of multivariate polynomials.

Lipson has also programmed the algorithm in the language "SCOPE FORMAX" [4].

1. The Fraction-Free Gaussian Elimination

Consider the n x (n+m) augmented matrix of a system of linear equations Ax = B:

$$A_{ij}(0) = (a_{ij}(0)) = (A|B)$$

Then the modified algorithm is given

$$a_{ij}^{(k)} = \frac{a_{kk}^{(k-1)} a_{ij}^{(k-1)} - a_{kj}^{(k-1)} a_{ik}^{(k-1)}}{a_{k-1,k-1}^{(k-2)}}$$

where

$$k = 1,2,\ldots, n-1$$

$$i = k+1,\ldots, n$$

$$j = k+1,\ldots, n+m$$

and with $A^{(0)}$ as given before, and where

$$a_{00}^{(-1)} = 1$$

$$a_{ij}^{(k)} = \begin{cases} a_{ij}^{(k-1)} & i = 1,\ldots,k; \quad j = 1,\ldots,n+m \\ 0 & i = k+1,\ldots,n; \quad j = 1,\ldots,k \end{cases}$$

Also, given here without proof (for the proof see Ref. 4) :

The determinant of the matrix A

$$DET(A) = a_{nn}^{(n-1)}$$

The formula given above was programmed using normal FORMAC, also ignoring for purpose of clarification any pivot element check for zero.

The algorithm alone does not give a fraction-free algebraic representation. Some processing is necessary to bring dividend and divisor into a form such that cancellation takes place.

A very useful method may be to use this algorithm to compute the determinant of a matrix with single letters as elements, followed by a substitution process, where more complicated expressions are substituted for the letters.

IV. APPLICATION OF FORMAC IN OTHER ALGEBRAIC PROBLEMS OF CONTROL THEORY

In this chapter some other applications of the algebraic manipulation language FORMAC in control theory are proposed. Some examples will be given, the programs being more of demonstration programs than operational programs.

The reason for this is mainly the lack of time which is needed to prepare an optimal program, optimal with respect to execution time and space requirement.

A. ROOT LOCUS THEORY

The impulse response of a linear time-invariant system is determined by a sum of exponential time functions. The exponents of these terms are the roots of the characteristic equation of the system.

A system is stable if and only if none of the n roots of the characteristic equation has positive real parts.

Consider the polynomial in the Laplace-variable S which represents a characteristic equation:

$$F(s) = \sum_{k=o}^{n} a_k s^k = 0$$

The coefficients a_k can be complex. Some of the coefficients can include a design parameter, K.

The polynomial F(s) can be partitioned into two parts as

$$F(s) + K\ B(s) \qquad (1)$$

The complex variable S is replaced as $s = \sigma + j\,\omega$. We require the real and imaginary parts of $F(\sigma, \omega)$ to be zero independently:

$$\text{Re } A(\sigma,\omega) + K \text{ Re } B(\sigma,\omega) = 0 \qquad (2a)$$

$$\text{Im } A(\sigma,\omega) + K \text{ Im } B(\sigma,\omega) = 0 \qquad (2b)$$

Using the set of simultaneous equations (2a) and (2b) the parameter K can be eliminated:

$$\text{Re } B * \text{Im } A - \text{Re } A * \text{Im } B = 0 \qquad (3)$$

This equation is the equation of the root locus; that is, every root of the characteristic equation $F(s) = 0$ must satisfy the equation (3).

In the computer program (Program 3) this equation is called $\text{RLOC} = f(\sigma,\omega)$. RLOC contains in every term the variable ω, because the real axis is always a locus for the roots. This factor of ω can be removed from RLOC.

If now a value for one of the variables σ or ω is assumed RLOC will give those points in the s-plane which satisfy equation (3). These points are on the root locus.

1. A FORMAC Program For the Root Locus

As an example for the application of FORMAC a program is given (Program 3).

The program reads in a polynomial CHEQ and the parameter PARAM in algebraic form. Then the polynomial is expanded and then, following the development as described before, the equation (3) of the root locus is derived in algebraic form.

At this point values for ω or σ in the range of interest can be chosen and the equation for the root locus can be evaluated, giving values for ω or σ respectively.

Normally the expression in the remaining unknown will not be linear. In that case the expression is treated as a polynomial, the roots of which can be found using a root-finding routine.

Finally the results can be plotted, giving the complete root locus in the range of interest.

To evaluate the values of the parameter at points along the root locus it is necessary to substitute back the values of σ and ω for those points into equation (2).

2. Discussion

The conventional root-locus procedure [9] is basically a trial-and-error method:

For a range of parameter values $K_1, K_2, \ldots, K_n$, for which one suspects an acceptable area of root locations in the s-plane, the roots of the characteristic equation are found by a root-finding method, probably a computer-programmed routine, using a standard method.

The the root locus is inspected in the area of interest for the values of σ and ω and the corresponding parameter values are selected.

The problem is that values of a quantity which are sought have to be assumed beforehand.

The method proposed here [10] avoids this sequence. The equation of the root locus is evaluated in the range of

interest for the variables σ and ω. Then by back-substitution of σ and ω into the given characteristic equation the parameter values are calculated. Another advantage is the fact that the degree of the root locus equation (3) is generally of lesser degree than the characteristic equation. This face results from possible cancellations of terms of equal powers in the process of parameter elimination. In example 1 the degree of the characteristic equation is 4 while the equation of the root locus is of degree 3 in σ and of degree 2 in ω.

B. FREQUENCY FUNCTIONS

Consider a transfer function P(s) = Y(s)+X(s). P(s) might contain some parameter in symbolic form. It is possible to use a FORMAC program to evaluate the absolute value of P(s) and the angle of P(s) as expressions containing the parameter in symbolic form. For complicated transfer functions it might be easier to evaluate the complex algebra only once and then vary the parameter to obtain values for magnitude and angle of the transfer function.

For the latter purpose it is mentioned that in FORMAC transcendental functions with symbolic arguments can be used.

C. LAPLACE TRANSFORMS AND INVERSION OF LAPLACE TRANSFORMS

Consider a differential equation in the time domain. If such an equation is given as input to a FORMAC program it is possible to perform a Laplace transform on the equation. The basic problem to be solved would be the replacement of one operator by another one. If both sides of the differential

equation would be read in as two separate FORMAC variables, a transfer function could be formed.

Another definite possibility is the analysis of an algebraic expression in the variable s and subsequent recognition of certain algebraic patterns, which can then be replaced by an expression in the time domain. This method would be similar to the "manual" approach, because one often solves these problems in the same way: recognizing the transform pair from experience (or tables).

D. SENSITIVITY ANALYSIS

A very powerful feature of FORMAC is its ability to form partial derivatives of user specified order. Also the mathematical functions such as sine, cosine, exponential, logarithm can be used with symbolic arguments.

Consider a mathematical model T(k), which might be a transfer function or frequency response function, of a linear time-invariant system.

The sensitivity of T(k), k being a variable, with respect to k is defined: [9]

$$S_k^{T(k)} = \frac{d \ln T(k)}{d \ln k} = \frac{d\, T(k)}{d\, k} \frac{k}{T(k)} .$$

The sensitivity of the phase angle of T(k), Φ_T with respect to parameter k is defined:

$$S_k^{\Phi_T} = \frac{d\, \Phi_T \; k}{d\, k\, \Phi_T} .$$

A FORMAC program which would find the derivatives involved in the sensitivity analysis could be time saving, if symbolic

arguments were involved and one would like to vary the parameter. A program for the finding of the derivatives would only be used once.

V. CONCLUSION

A method of solution of a pattern recognition problem in the analysis of a Mason flow graph has been shown. It also has been shown that computer programs, using an algebraic manipulation language FORMAC can be used in the subsequent evaluation of the results of the pattern recognition problem. It is possible to evaluate in symbolic form certain system functions, for example the transfer function between specified nodes and the characteristic equation of large systems.

A FORMAC program can also be used to advantage in a modified root locus method.

It is felt that the application of an algebraic manipulation language (FORMAC) which has a wide range of capabilities as compared to other more restricted languages which may handle only polynomials, is of great value in the solution of algebraic problems in control theory and in network theory.

Of particular importance is the fact that FORMAC is a superset of PL/1. All the capabilities of PL 1 are usable together with FORMAC.

COMPUTER PROGRAMS

1. Program DTERM, a FORMAC program

   Purpose: To evaluate a determinant of a square matrix of order up to 9. The elements of the matrix are univariate polynomials with symbolic coefficients. Given is one example.

2. Program FINDLOOP, a PL/1 program

   Purpose: To evaluate a control vector LIST1(4) which is needed for the Mason flow graph analysis. Given is one example.

3. Program MASON_1, a FORMAC program

   Purpose: To find sets of touching loops from given topological matrices describing a Mason flow graph and to evaluate the graph determinant. Given is one example.

4. Program MASON_2, a FORMAC program

   Purpose: To evaluate the cofactors in the Mason gain formula. Given is one example.

5. Program RTLOC, a FORMAC program

   Purpose: To find the equation of the root locus from a given characteristic equation in symbolic form. Given are three examples.

```
//JOBLIB   DD  DISP=SHR,DSNAME=SYS1.FORMAC
//        EXEC  FORMAC
//FORMAC.SYSIN DD  *
 TEST:PROCEDURE OPTIONS(MAIN):
      FORMAC_OPTIONS:
      OPTSET(LINELENGTH=72):
   /*


   */
  /* THIS PROGRAM USES INPUT IN FORM OF AN INPUT MATRIX A.
   THIS INPUT MATRIX IS A CHARACTER ARRAY OF VARYING LENGTH.
   EACH ELEMENT A(I,J) IS A POLYNOMIAL IN THE VARIABLE S.
   THE POLYNOMIAL MUST BE SPECIFIED IN THIS FORM:
              A(I,J) = 'A + B*S +C*S**2 QD*S**3....7*S**N'
   NOTE THE ' AFTER THE = SIGN AND AT THE END OF THE INPUT.
   THE ROW- AND COLUMN INDICES I,J CAN BE IN ANY ORDER.
   IF A SPECIFIC ELEMENT A(A,B) = 0 THEN IT NEAD NOT BE SPECIFIED.
   AT THE END OF THE LAST ELEMENT A : HAS TO APPEAR.EXAMPLE:
              A(3,3) = 'A +B*(K-G*F**2)*S +(15.55*KL+G*H)*S**5':
   COMPUTATION TIME AND SPACE CAN BE SAVED IF THE INPUT IS IN THE FORM
   DESCRIBED ABOVE. IF  THE PROBLEM IS RELATIVE SMALL THE TERM
   PDD SHOULD BE EXPANDED USING THE STATEMENT:
              LET(PDD=EXPAND("A(M,IP(M))") ):
   IF THIS STATEMENT IS INSERTED  FOR THE STATEMENT:
              LET(PDD="A(M,IP(M))" ):
   THE INPUT NEED NOT TO BE IN THE FORM OF A POLYNOMIAL. EXAMPLE:
              A(2,1) = '(A*S-B*S**2)*(K-R*L-F*S+S**5)/(K*S+15)'
   IN THE INPUT POLYNOMIAL NEGATIVE EXPONENTS OF THE VARIABLE S
   ARE ALLOWED. THE OUTPUT FROM THE PROGRAM  ARE THE COEFFICIENTS
   C(I) IN THE EXPANDED DETERMINANT DET(A), WHICH IS A POLYNOMIAL IN S.

      PART  1

  */
      DCL (NARG1,NARG2) DEC FIXED:
      DCL IP(N) DEC FIXED CTL:
      DCL ((P,D)(2:N)) DEC FIXED CTL:
      DCL (K,Q,T) DEC FIXED:
      DCL FIRST BIT(1):
      DCL A(N1,N1) CHARACTER(75) VARYING CTL:
      DCL PDD CHARACTER(75) VARYING:
      DCL SGN DEC FIXED:
      DCL MXPOW DEC FIXED:
      DCL NF DEC FIXED:
```

```
   /*

      PART  2

   */
      GET DATA(N1);
      GET DATA(NF);
      N=N1;
ALO1:ALLOCATE IP,P,D,A;
      A='0';
      GET DATA (A);
      MXPOW=0;
      N2=0;
      DO I=1 TO N;
      N2=N2+1;
      IP(I)=N2;
      END;
      FIRST='1'B;
      IN=0;
   /*

      PART  3

   */
      DO I=1 TO NF;
      SGN=(-1.)**I;
      CALL PERM;
      LET(PD="SGN");
      DO M=1 TO N1;
      IF A(M,IP(M))='0' THEN GO TO BROW;
      END;
   /*

      PART  4

   */
      IN=IN+1;
      DO M=1 TO N1;
      LET(PDD="A(M,IP(M))");
      IF M=1 THEN DO; LET(PD=PD*PDD); GO TO CONT;END;
      LET(P5=0);
      NARG1=NARGS(PDD);
```

```
      NARG2=NARGS(PD);
      IF LOP(PDD)=26 THEN NARG1=1;
      IF LOP(PD)=26 THEN NARG2=1;
      IF NARG1=0 THEN NARG1=1;
      IF NARG2=0 THEN NARG2=1;
      DO J=1 TO NARG1;
      IF NARG1=1 THEN LET(P1=PDD);
      ELSE LET(P1=ARG("J",PDD));
      DO L=1 TO NARG2;
      IF NARG2=1 THEN LET(P2=PD*P1);
      ELSE LET(P2=ARG("L",PD)*P1);
      LET(POW=LOWPOW(P2,S));
      IF INTEGER(POW)=0
         THEN DO;LET(P5=P5+P2);ATOMIZE(P2);SAVE(P5);END;
         ELSE DO; LET(P6=COEFF(P5,S**POW));
                  IF LOP(P6)=36 THEN DO;LET(P5=P5+P2);ATOMIZE(P2:P6);
                                        SAVE(P5);END;
                  ELSE DO;LET(P2=P2/(S**POW);
                                P7=CHAIN(P6*S**POW,(P6+P2)*S**POW));
                           ATOMIZE(P6;P2);
                            LET(P5=REPLACE(P5,P7));
              END;END;
      ATOMIZE(P7);
      END;
      END;
      LET(PD=P5);
      ATOMIZE(P1;P5);
CONT:;
      END;
   /*

      PART  5

   */
PUT EDIT('EXPANDED TERM NO.',IN,'OF THE DETERMINANT DET(A)')(SKIP(2),
A);
      LET(POW=HIGHPOW(PD,S));
      IF INTEGER(POW)>MXPOW THEN MXPOW=INTEGER(POW);
      PUT LIST('COEFFICIENTS OF THE VARIABLE IN ABOVE TERM PD');
      DO M=1 TO INTEGER(POW);
      LET(P1=COEFF(PD,S**"M")*S**"M");
      IF LOP(P1)=36 THEN GO TO FIN;
      LET(PD=REPLACE(PD,P1,0));
      SAVE(PD);
      LET(P1=P1/S**"M");
      LET(C("IN","M")=EXPAND(P1));
```

```
      ATOMIZE(P1):
FIN:  ;
      PRINT_OUT(VARBL=S**"M"):
      PRINT_OUT(C("IN","M")):
      SAVE(C("IN","M")):
      END:
   /*

      PART  6

   */
      IF LOP(PD)=36 THEN GO TO FIN2:
      LET(C("IN",0)=EXPAND(PD)):
      PRINT_OUT(C("IN",0)):
      SAVE(C("IN",0)):
FIN2:;
      ATOMIZE(PD):
RROW:;
      END:
   /*

      PART  7

   */
      PUT LIST('DETERMINANT IS EXPANDED.'):
      PUT LIST('THE COEF(I) IN THE COMPLETE EXPANSION OF DET(A) ARE:'):
      DO L=0 TO MXPOW:
      LET(L="L"):
      DO M=1 TO IN:
      LET(M="M"):
      IF LOP(C( M , L ))=46 THEN GO TO FIN5:
      IF LOP(COEF( L ))=46 THEN DO:LET(COEF( L )=C( M , L )): END:
                           ELSE LET(COEF( L )=COEF( L )+C( M , L )):
      ATOMIZE(C( M , L )):
FIN5:;
      END:
      PRINT_OUT(COEF( L )):
      ATOMIZE(COEF( L )):
      END:
   /*

      PART  8
```

```
   */
 PERM:PROCEDURE;
      N=N1;
      IF FIRST THEN INIT:DO;
         P=0;D=1;FIRST='0'B;
         END INIT;
      K=0;
 IND :P(N),Q=P(N)+D(N);
      IF Q=N THEN DO;
         D(N)=-1.;
         GO TO LOOP;
         END;
      IF Q¬=0 THEN GO TO TRAS;
      D(N)=1; K=K+1;
 LOOP:IF N>2 THEN DO;
         N=N-1;
         GO TO IND;
         END;
      Q=1;
      FIRST='1'B;
 TRAS:Q=Q+K;
      T=IP(Q);
      IP(Q)=IP(Q+1);
      IP(Q+1)=T;
 END PERM;
 END TEST;
//GO.SYSIN  DD  *
N1=4;
NF=24;
A(1,1)='S'
A(1,2)='-1'
A(2,2)='(JO*(RO1*RO2)**2+JM)*S+(FO*(RO1*RO2)**2+FM)'
A(2,3)=' -(RO1*RO2*KT*IA)'
A(3,1)='KA*KPO'
A(3,2)='KA*H*1/RO2'
A(3,3)='L*S+RF'
A(3,4)='KA*(B-A)'
A(4,1)='KPO'
A(4,4)='S+(A+B)';
```

```
INPUT MATRIX A

A(1,1)='S'                  A(1,2)='-1'                 A(1,3)='0'
A(1,4)='0'                  A(2,1)='0'                  A(2,2)='(JO*(RO1*RO2)**2
+JM)*S+(FO*(RO1*RO2)**2+FM)'                           A(2,3)='-(RO1*RO2*KT*IA)'
A(2,4)='0'                  A(3,1)='KA*KPO'             A(3,2)='KA*H*1/RO2'
A(3,3)='L*S+RF'             A(3,4)='KA*(B-A)'           A(4,1)='KPO'
A(4,2)='0'                  A(4,3)='0'                  A(4,4)='S+(A+B)';
```

EXPANDED TERM NO. 1 OF THE DETERMINANT DET(A).
COEFFICIENTS OF THE VARIABLE IN THE TERM:

VARBL = S

C(1,1) = IA KT RO2 RO1 KPO KA

C(1,0) = IA KT RO2 RO1 A KPO KA + IA KT RO2 RO1 B KPO KA

EXPANDED TERM NO. 2 OF THE DETERMINANT DET(A).
COEFFICIENTS OF THE VARIABLE IN THE TERM:

C(2,0) = IA KT RO2 RO1 A KPO KA - IA KT RO2 RO1 B KPO KA

EXPANDED TERM NO. 3 OF THE DETERMINANT DET(A).
COEFFICIENTS OF THE VARIABLE IN THE TERM:

VARBL = S

C(3,1) = IA KT RO1 A KA H + IA KT RO1 B KA H

VARBL = S^2

C(3,2) = IA KT RO1 KA H

EXPANDED TERM NO. 4 OF THE DETERMINANT DET(A).
COEFFICIENTS OF THE VARIABLE IN THE TERM:

VARBL = S

$$C(4,1) = RO2^2\ RO1^2\ A\ FO\ RF + RO2^2\ RO1^2\ B\ FO\ RF + A\ FM\ RF + B\ FM\ RF$$

VARBL = S^2

C(4,2) = $RO2^2$ $RO1^2$ A JO RF + $RO2^2$ $RO1^2$ B JO RF + A JM RF + B JM RF + $RO2^2$ $RO1^2$ FO RF + FM RF + $RO2^2$ $RO1^2$ A L FO + $RO2^2$ $RO1^2$ B L FO + A L FM + B L FM

VARBL = S^3

C(4,3) = $RO2^2$ $RO1^2$ JO RF + JM RF + $RO2^2$ $RO1^2$ A L JO + $RO2^2$ $RO1^2$ B L JO + A L JM + B L JM + $RO2^2$ $RO1^2$ L FO + L FM

VARBL = S^4

C(4,4) = $RO2^2$ $RO1^2$ L JO + L JM

DETERMINANT IS EXPANDED.
THE COEFFICIENTS COEF(I) IN THE COMPLETE EXPANSION OF DET(A) ARE:

COEF(0) = 2 IA KT RO2 RO1 A KPO KA

COEF(1) = $RO2^2$ $RO1^2$ A FO RF + $RO2^2$ $RO1^2$ B FO RF + A FM RF + B FM RF + IA KT RO1 A KA H + IA KT RO1 B KA H + IA KT RO2 RO1 KPO KA

COEF(2) = $RO2^2$ $RO1^2$ A JO RF + $RO2^2$ $RO1^2$ B JO RF + A JM RF + B JM RF + $RO2^2$ $RO1^2$ FO RF + FM RF + IA KT RO1 KA H + $RO2^2$ $RO1^2$ A L FO + $RO2^2$ $RO1^2$ B L FO + A L FM + B L FM

COEF(3) = $RO2^2$ $RO1^2$ JO RF + JM RF + $RO2^2$ $RO1^2$ A L JO + $RO2^2$ $RO1^2$ B

$$\text{COEF}(4) = \frac{L\,JO + A\,L\,JM + B\,L\,JM + RO2^2\ RO1^2\ L\,=O + L\,FM}{RO2^2\ RO1^2\ L\,JO + L\,JM}$$

```
//         EXEC  PL1CLG
//PL1L.SYSIN DD *
  /*  THIS PROGRAM PRODUCES THE CONTROL VECTOR LIST1(4).
      IN THIS TEST ROUTINE THE FOLLOWING NAMES ARE USED:

         1) FOR LIST1(4) : NODEFLAG
         2) FOR LIST1(1) : W2

      A SET OF TEST DATA IS GIVEN.ALSO SHOWN IS THE PRINTOUT.
  */
 TEST:PROCEDURE OPTIONS (MAIN);
      DCL (FIND ,N1,NODES) DEC(2) FIXED;
      DCL NODEFLAG(N1) BIN(1) FIXED CTL;
      DCL  W2(N1) DEC(2) FIXED CTL;
      CALL FINDLOOP;
 FINDLOOP: PROCEDURE;
      GET EDIT (N1,NODES) (SKIP(1),F(2),X(1),F(2));
 AL01:ALLOCATE W2,NODEFLAG;
      NODEFLAG = 1;
      GET EDIT (W2) ((N1) (SKIP(1),F(2),X(1)));
      DO I=1 TO NODES;
      FIND=I;
      DO K=1 TO N1;
      IX=INDEX(W2(K),FIND);
      IF IX=0 THEN GO TO FDL3;ELSE IX=K;GOTO FDL2;
 FDL3:;
      END;
 FDL2:;
      IF IX=0 THEN GO TO FDL1;
      IF IX=N1
          THEN NODEFLAG (IX)=0;
          ELSE  IF W2(IX+1)=W2(IX) THEN;
                ELSE NODEFLAG(IX)=0;
      IF IX=1 THEN;
              ELSE NODEFLAG(IX-1)=0;
 FDL1:IF I=NODES THEN IF W2(N1) = W2(N1-1)
                      THEN NODEFLAG(N1)=0;
                      ELSE;
      ELSE;
      END;
      PUT EDIT ('NODES') (PAGE,LINE(2),A(9));
      PUT EDIT (W2) ((N1) (F(2),X(1)));
      PUT EDIT ('NODEFLAGS') (LINE(5),A(9));
      PUT EDIT (NODEFLAG) ((N1) (X(1),F(1),X(1)));
      END FINDLOOP;
      END TEST;
  /*  IT FOLLOWS TEST DATA INPUT..ON THE FIRST DATA CARD
      ARE TWO NUMBERS: 1) THE NUMBER OF NODES IN LIST1(1) AND
```

```
                        2) THE HIGHEST NODE NUMBER
      THE FOLLOWING NUMBERS ARE THE ACTUAL NODES AS THEY WOULD
      APPEAR IN THE INPUT LIST LIST1(1).
   */
//GO.SYSPRINT DD SYSOUT=B,DCB=(RECFM=FB,LRECL=80,BLKSIZE=80)
//GO.SYSIN   DD *
THIS CARD IS A BLANK CARD. IT MUST BE USED WITH TEST DATA.
32 30
01
01
02
02
02
03
03
04
05
05
06
06
06
06
06
07
07
08
09
09
10
10
11
12
12
15
16
16
26
26
26
30
```

FINDLOOP Output

```
NODES    1  1  2  2  2  2  3  3  4  5  5  5  5  6  6  6  6  7  7  8  9  9 10 10 11 11
 2 12 15 16 16 26 26 26 30
NODEFLAGS 1  0  1  1  0  1  0  1  0  0  1  0  1  1  1  1  0  1  0  0  1  0  1  0  0  0
 1  0  0  1  0  1  1  0  0
```

```
//JOBLIB   DD  DISP=SHR,DSNAME=SYS1.FORMAC
//  EXEC  FORMAC
//FORMAC.SYSIN DD  *
 MASON_1:PROCEDURE OPTIONS(MAIN);
         FORMAC OPTIONS;
         OPTSET(LINELENGTH=72);
         DCL  LT(NL,NL) DEC FIXED CTL;
         DCL  INX(NT,NT) DEC FIXED CTL;
         GET DATA(NL);GET DATA(ND);GET DATA(NN);
   AL1:  ALLOCATE LT;LT=0;
         GET DATA(LT);
         NT=SUM(LT)/2;
   AL2:  ALLOCATE INX;
         IX=1;
         DO K=1 TO NL-1;
         DO L=K+1 TO NL;
         IF LT(K,L)=0 THEN GO TO FIN;
         INX(IX,1)=K; INX(IX,2)=L; IX=IX+1;
   FIN:  END;
         PUT SKIP(15);
         PUT LIST('SETS OF TOUCHING LOOPS LOOP2(I):');PUT SKIP(1);
         LET(DELTA=1);
         DO K=1 TO NL;
         LET(DELTA=EXPAND(DELTA*(1-L("K"))));
         DO L=1 TO IX-1;
         LET(P1=L("INX(L,1)")*L("INX(L,2)"));
         IF K=1 THEN PRINT OUT(LOOP2("L")=P1);
         LET(DELTA=REPLACE(DELTA,P1,0));
         END;
         END;
         PRINT OUT(DELTA);
 END MASON_1;
//GO.SYSPRINT DD SYSOUT=B,DCB=(RECFM=FB,LRECL=80,BLKSIZE=80)
//GO.SYSIN  DD  *
NL=4;
ND=3;
NN=7;
LT(2,3)=1  LT(3,2)=1  LT(3,4)=1  LT(4,3)=1;
```

```
SETS OF TOUCHING LOOPS LOOP2(I):
LOOP2(1) = L(3) L(2)
-------------------
LOOP2(2) = L(4) L(3)
-------------------
DELTA =  - L(1) - L(2) - L(3) - L(4) + L(2) L(1) - L(4) L(2) L(1) + L(3)
  L(1) + L(4) L(1) + L(4) L(2) + 1
-------------------------------------
```

```
//JOBLIB   DD  DISP=SHR,DSNAME=SYS1.FORMAC
//        EXEC  FORMAC
//FORMAC.SYSIN DD  *
 MASON_2:PROCEDURE OPTIONS(MAIN);
        FORMAC OPTIONS;
        OPTSET(LINELENGTH=72);
        DCL PL(NL,NP) DEC FIXED CTL;
        DCL P(NP,NN) DEC FIXED CTL;
        GET DATA(NL);GET DATA(NP);GET DATA(NN);
        ALLOCATE PL,P;
        P=0;PL=0;
        GET DATA(PL);
        GET DATA(P);
        LET(DELTA=-L(1)-L(2)-L(3)-L(4)+L(2)*L(1));
        LET(DELTA=DELTA+L(3)*L(1)-L(4)*L(2)*L(1)+L(4)*L(1)+L(4)*L(2)+1);
        DO K=1 TO NP;
        LET(CHN=CHAIN(L(0),0));
        DO L=1 TO NL;
        IF PL(L,K)=0 THEN GO TO FIN2;
        LET(CHN=CHAIN(CHN,L("L"),0));
FIN2:;
        END;
        LET(DELTK("K")=REPLACE(DELTA,CHN));
        PUT EDIT('PATH ',K,' IS TOUCHED BY LOOPS WHICH ARE REPLACED AS:')
                (A,F(2),A);PUT SKIP(2);
        PRINT_OUT(CHN);
        PRINT_OUT(DELTK("K"));
        PUT SKIP(3);
        END;
 END MASON_2;
//GO.SYSPRINT DD SYSOUT=B,DCB=(RECFM=FB,LRECL=80,BLKSIZE=80)
//GO.SYSIN  DD  *
NL=4;
NP=3;
NN=7;
PL(1,1)=1 PL(1,2)=1 PL(1,3)=1 PL(2,1)=1 PL(3,1)=1 PL(3,2)=1 PL(3,3)=1
PL(4,1)=1 PL(4,2)=1 PL(4,3)=1 ;
P(1,1)=1 P(1,2)=1 P(1,7)=1 P(2,1)=1 P(2,2)=1 P(2,6)=1 P(2,7)=1 P(3,1)=1
P(3,2)=1 P(3,3)=1 P(3,4)=1 P(3,5)=1 P(3,6)=1 P(3,7)=1 ;
```

```
PATH  1 IS TOUCHED BY LOOPS WHICH ARE REPLACED AS:
CHN = CHAIN ( L(0), 0, L(1), 0, L(2), 0, L(3), 0, L(4), 0 )
-----------------------------------------------------------
DELTK(1) = 1
------------

PATH  2 IS TOUCHED BY LOOPS WHICH ARE REPLACED AS:
CHN = CHAIN ( L(0), 0, L(1), 0, L(3), 0, L(4), 0 )
---------------------------------------------------
DELTK(2) =  - L(2) + 1
----------------------

PATH  3 IS TOUCHED BY LOOPS WHICH ARE REPLACED AS:
CHN = CHAIN ( L(0), 0, L(1), 0, L(3), 0, L(4), 0 )
---------------------------------------------------
DELTK(3) =  - L(2) + 1
----------------------
```

```
//JOBLIB    DD  DISP=SHR,DSNAME=SYS1.FORMAC
//         EXEC  FORMAC,PARM.PL1L='SIZE=999999,SM=(2,80),LIST',          *
//               PARM.LKED='MAP,LIST',REGION.GO=200K
//FORMAC.SYSIN DD  *
 RTLOC:PROCEDURE OPTIONS(MAIN);
       FORMAC_OPTIONS;
       OPTSET(LINELENGTH=72);
       DCL INX DEC FIXED;
       DCL CHEQ CHARACTER(75) VARYING;
       DCL PARAM CHARACTER(75) VARYING;
 INPUT:;
       GET DATA(CHEQ); GET DATA(PARAM);
       ON ENDFILE(SYSIN) GO TO  FIN;
       LET(CHEQ="CHEQ");
       LET(PA="PARAM");
       LET(CHEQ=EXPAND(CHEQ));
       LET(CQ=EXPAND(EVAL(CHEQ,S,SIGMA+#I*OMEGA));
       BB=COEFF(CQ,PA);
       AA=EXPAND(CQ-PA*BB);
       IA=COEFF(AA,#I);
       IB=COEFF(BB,#I);
       RA=EXPAND(AA-IA*#I);
       RB=EXPAND(BB-IB*#I));
       LET(VARBL=OMEGA);
       DO I=1 TO 2;
       LET(RLOC=EXPAND(RB*IA-RA*IB);
       RLOC=EXPAND(RLOC/OMEGA));
       PUT SKIP(2);
       LET(POW=HIGHPOW(RLOC,VARBL));
       PRINT_OUT(CHEQ);
       PRINT_OUT(CQ);
       PRINT_OUT(VARBL);
       PRINT_OUT(PARAM=PA);
       PRINT_OUT(RLOC);
       DO L=I TO INTEGER(POW);
       LET(COF=COEFF(RLOC,VARBL**"L"));
       LET(CO("L")=COF);
       IF LOP(COF) =36 THEN GO TO L1;
       LET(RLOC=EXPAND(RLOC-COF*VARBL**"L"));
  L1:; END;
       LET(CO(0)=RLOC);
  L2:;
       DO L=0 TO INTEGER(POW);
       LET(COFF=CO("L"));
       PRINT_OUT(CO("L"));
       ATOMIZE(CO("L"));
       END;
       LET(VARBL=SIGMA);
```

```
      END;
      GO TO INPUT;
 FIN:;
 END RTLOC;
//GO.SYSPRINT DD SYSOUT=B,DCB=(RECFM=FB,LRECL=80,BLKSIZE=80)
//GO.SYSIN  DD  *
 CHEQ='(S+P)*(S**2 +2*PSI*OMEGN*S +OMEGN**2) +K';
PARAM='K';
 CHEQ='S**4 +8*S**3 +17*S**2 +10*S +K';
PARAM='K';
 CHEQ='S**8 + 8*S**3 + K +B*S**3+C*S**9-(D+F/J)*S**6';
PARAM='K';
```

$$CHEQ = \frac{K + 2\,OMEGN\,PSI\,P\,S + OMEGN^2\,S^2 + OMEGN^2\,P + P\,S^2 + 2\,OMEGN\,PSI\,S^2 + S^3}{}$$

$$CQ = \frac{K + 2\,SIGMA\,OMEGN\,PSI\,P + 2\,\#I\,OMEGA\,OMEGN\,PSI\,P + 2\,\#I\,OMEGA\,SIGMA\,P + OMEGN^2\,P + SIGMA^2\,P - OMEGA^2\,P + 4\,\#I\,OMEGA\,SIGMA\,OMEGN\,PSI + 2\,SIGMA^2\,OMEGN\,PSI - 2\,OMEGA^2\,OMEGN\,PSI - 3\,OMEGA^2\,SIGMA + SIGMA\,OMEGN^2 + \#I\,OMEGA\,OMEGN^2 + 3\,\#I\,OMEGA\,SIGMA^2 - \#I\,OMEGA^3 + SIGMA^3}{}$$

$$PARAM = K$$

$$RLOC = \frac{2\,OMEGN\,PSI\,P + 2\,SIGMA\,P + 4\,SIGMA\,OMEGN\,PSI + OMEGN^2 + 3\,SIGMA^2 - OMEGA^2}{}$$

VARBL = OMEGA

$$CO(0) = \frac{2\ OMEGN\ PSI\ P + 2\ SIGMA\ P + 4\ SIGMA\ OMEGN\ PSI + OMEGN^2 + 3}{SIGMA^2}$$

$$CO(1) = 0$$

$$CO(2) = -1$$

VARBL = SIGMA

$$CO(0) = \frac{2\ OMEGN\ PSI\ P + OMEGN^2 - OMEGA^2}{2\ P + 4\ OMEGN\ PSI}$$

$$CO(2) = 3$$

$$CHEQ = \frac{10\,S + K + 17\,S^2 + 8\,S^3 + S^4}{}$$

$$CQ = \frac{K + 10\,SIGMA + 34\,\#I\,OMEGA\,SIGMA - 24\,OMEGA^2\,SIGMA - 4\,\#I\,OMEGA^3}{SIGMA + 10\,\#I\,OMEGA + 24\,\#I\,OMEGA\,SIGMA^2 - 6\,OMEGA^2\,SIGMA^2 + 4\,\#I}$$

$$OMEGA\,SIGMA^3 - 8\,\#I\,OMEGA^3 + 17\,SIGMA^2 + 8\,SIGMA^3 + SIGMA^4 - 17$$

$$OMEGA^2 + OMEGA^4$$

$$PARAM = K$$

$$RLOC = \frac{34\,SIGMA - 4\,OMEGA^2\,SIGMA + 24\,SIGMA^2 + 4\,SIGMA^3 - 8\,OMEGA^2}{+ 10}$$

VARBL = OMEGA

$$CO(0) = 34\ SIGMA + 24\ SIGMA^2 + 4\ SIGMA^3 + 10$$

$$CO(1) = 0$$

$$CO(2) = -4\ SIGMA - 8$$

VARBL = SIGMA

$$CO(0) = -8\ OMEGA^2 + 10$$

$$CO(1) = -4\ OMEGA^2 + 34$$

$$CO(2) = 24$$

$$CO(3) = 4$$

```
              6           3       9       6         3     8
CHEQ = K - S   F /  J + S   B + S   C - S   D + 8 S   + S
------------------------------------------------------------
                      5                    4        2
CO = K - 6 #I OMEGA    SIGMA F /  J - 15 OMEGA   SIGMA   F /  J + 20 #I
------------------------------------------------------------------------
     3        3                    2        4                        5
OMEGA   SIGMA   F /  J + 15 OMEGA   SIGMA   F /  J - 6 #I OMEGA SIGMA
------------------------------------------------------------------------
             6             6                  2
F /  J - SIGMA   F /  J + OMEGA   F /  J - 3 OMEGA   SIGMA B + 3 #I
------------------------------------------------------------------------
              2         3              3             8
OMEGA SIGMA    B + SIGMA   B - #I OMEGA   B + 9 OMEGA   SIGMA C - 36 #I
------------------------------------------------------------------------
     7      2             6      3                 5      4
OMEGA  SIGMA   C - 84 OMEGA  SIGMA   C + 126 #I OMEGA  SIGMA   C +
------------------------------------------------------------------------
          4      5                3      6              2      7
126 OMEGA  SIGMA   C - 84 #I OMEGA  SIGMA   C - 36 OMEGA  SIGMA   C
------------------------------------------------------------------------
                    8         9             9                 5
 + 9 #I OMEGA SIGMA   C + SIGMA   C + #I OMEGA   C - 6 #I OMEGA   SIGMA
------------------------------------------------------------------------
            4      2                3      3             2      4
D - 15 OMEGA  SIGMA   D + 20 #I OMEGA  SIGMA   D + 15 OMEGA  SIGMA
------------------------------------------------------------------------
                    5        6          6             2
D - 6 #I OMEGA SIGMA   D - SIGMA  D + OMEGA   D - 24 OMEGA   SIGMA - 8
------------------------------------------------------------------------
        7                            2           6      2
#I OMEGA   SIGMA + 24 #I OMEGA SIGMA   - 28 OMEGA  SIGMA   + 56 #I
------------------------------------------------------------------------
     5      3            4      4              3      5           2
OMEGA  SIGMA   + 70 OMEGA  SIGMA   - 56 #I OMEGA  SIGMA   - 28 OMEGA
------------------------------------------------------------------------
     6                     7              3           3        8
SIGMA   + 8 #I OMEGA SIGMA   - 8 #I OMEGA   + 8 SIGMA   + SIGMA   +
------------------------------------------------------------------------
     8
OMEGA
-------
PARAM = K
---------
                 4                        2      3                  5
RLOC =  - 6 OMEGA   SIGMA F /  J + 20 OMEGA  SIGMA   F /  J - 6 SIGMA
------------------------------------------------------------------------
```

RTLOC Output, Example 3

$$F / J + 3\,SIGMA^2\,B - OMEGA^2\,B - 35\,OMEGA^6\,SIGMA^2\,C + 126\,OMEGA^4$$

$$SIGMA^4\,C - 84\,OMEGA^2\,SIGMA^6\,C + 9\,SIGMA^8\,C + OMEGA^8\,C - 6\,OMEGA^4$$

$$SIGMA\,D + 20\,OMEGA^2\,SIGMA^3\,D - 6\,SIGMA^5\,D - 8\,OMEGA^6\,SIGMA + 56$$

$$OMEGA^4\,SIGMA^3 - 56\,OMEGA^2\,SIGMA^5 + 24\,SIGMA^2 + 8\,SIGMA^7 - 8\,OMEGA^2$$

VARBL = OMEGA

$$CO(0) = \frac{-6\,SIGMA^{5}\,E / J + 3\,SIGMA^{2}\,B + 9\,SIGMA^{8}\,C - 6\,SIGMA^{5}\,D + 24\,SIGMA^{2} + 8\,SIGMA^{7}}{}$$

$$CO(1) = 0$$

$$CO(2) = \frac{-8 + 20\,SIGMA^{3}\,E / J - 84\,SIGMA^{6}\,C + 20\,SIGMA^{3}\,D - 56\,SIGMA^{5} - 8}{}$$

$$CO(3) = 0$$

$$CO(4) = \frac{-6\,SIGMA\,E / J + 126\,SIGMA^{4}\,C - 6\,SIGMA\,D + 56\,SIGMA^{2}}{}$$

$$CO(5) = 0$$

$$CO(6) = \frac{-8\,SIGMA - 36\,SIGMA^{2}\,C}{}$$

$$CO(7) = 0$$

$$CO(8) = C$$

```
VARBL = SIGMA
-------------

CO(0) =  - OMEGA   B + OMEGA   C - 8 OMEGA
-----------------------------------------

                 4                      6
CO(1) =  - 6 OMEGA   F / J - 6 OMEGA   D - 8 OMEGA
-----------------------------------------------------

                 6
CO(2) = 3 B - 36 OMEGA   C + 24
-------------------------------

                 2                      4
CO(3) = 20 OMEGA   F / J + 20 OMEGA   D + 56 OMEGA
---------------------------------------------------

                 4
CO(4) = 126 OMEGA   C
---------------------

                 2                    2
CO(5) =  - 6 D - 6 F / J - 56 OMEGA
-------------------------------------

                 2
CO(6) =  - 84 OMEGA   C
-----------------------

CO(7) = 8
---------

CO(8) = 5 C
-----------
```

BIBLIOGRAPHY

1. Ward, John R., and Strum, Robert D., The Signal Flow Graph in Linear Systems Analysis, A programmed text, Prentice-Hall, Inc., 1968.

2. Chan, S. P., Introductory Topological Analysis of Electrical Networks, p. 246-275, Holt, Rinehart & Winston, Inc., 1969.

3. Dunn, William R. and Chan, Shu-Park, "Flowgraph Analysis of Linear Systems Using Remote Time-shared Computation", Journal of the Franklin Institute, Vol. 288, no. 5, p. 337-349, November 1969.

4. IBM Boston Programming Center, Programming Laboratory Report, Proceedings of the 1968 Summer Institute on Symbolic Mathematical Computation, ed. by Robert G. Tobey, June 1969.

5. Sammet, J. E., and Bond, E. R., "Introduction to FORMAC", IEEE Transactions on Electronic Computers, p. 386-394, August 1964.

6. IBM Corporation, 360D-03.3.004, PL/1 - FORMAC Symbolic Mathematics Interpreter, by R. Tobey, and others, September 1969.

7. Tobey, Robert G., "Eliminating Monotonous Mathematics with FORMAC", Communications of the ACM, Vol. 9, No. 10, p. 742-751, October 1966.

8. Sammet, Jean E., "Formula Manipulation Compiler", Datamation, p. 32-35, July 1966.

9. DiStefano, J. J., Stubberud, Allen R., and Williams, Ivan J., Theory and Problems of Feedback and Control Systems, Schaums Outline Series McGraw-Hill Company, 1967.

10. Thaler, G. F., Root Locus, Unpublished notes, Post-graduate School, Monterey, April 1970.

11. Russel, L., "Linear Circuit Analysis by Symbolic Algebra", Proceedings of 24th National Conference, Association for Computing Machinery, ACM Publication P-69, p. 573-586, 1969.

12. Gey, F. C., and Lesser, M. B., "Computer Generation of Series and Rational Function Solutions to Partial Differential Initial Value Problems", Proceedings of 24th National Conference, Association for Computing Machinery, ACM Publication P-69, p. 559-564, 1969.

INITIAL DISTRIBUTION LIST

		No Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Inspektion des Erziehungs- und Bildungswesens der Marine 294 Wilhelmshaven, Germany	1
4.	Dokumentationszentrale der Bundeswehr [See] 53 Bonn Friedrich - Ebert - Allee 34, Germany	1
5.	Professor George J. Thaler Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	15
6.	Kapitänleutnant Peter Merzhäuser 294 Wilhelmshaven, Germany Kommando Marineführungssysteme	2
7.	Mr. Wei Mun Syn Bldg. 014 International Business Machine Corp. Monterey and Cottle Roads San Jose, California 95114	1
8.	Dr. Ken Deckert International Business Machine Corp. Monterey and Cottle Roads San Jose, California 95114	1
9.	Mr. Martin Dost International Business Machine Corp. Monterey and Cottle Roads San Jose, California 95114	1
10.	Dr. Jean Sammet International Business Machine Corp. Cambridge, Massachusetts 02139	1

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

ORIGINATING ACTIVITY (Corporate author)	2a. REPORT SECURITY CLASSIFICATION
Naval Postgraduate School Monterey, California 93940	Unclassified
	2b. GROUP

REPORT TITLE

Symbolic Algebraic Manipulation By Digital Computer In Problems Of Control Theory

DESCRIPTIVE NOTES (Type of report and, inclusive dates)

Master's Thesis; June 1970

AUTHOR(S) (First name, middle initial, last name)

Klaus Peter Merzhäuser

REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
June 1970	94	12

a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO.	
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	

0. DISTRIBUTION STATEMENT

This document has been approved for public release and sale; its distribution is unlimited.

1. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Naval Postgraduate School Monterey, California 93940

3. ABSTRACT

A method of digital computer oriented analysis of a Mason flow graph is presented. Using this method in connection with a computer program implementing the algebraic manipulation language FORMAC can be a tool in expanding large determinants with symbolic polynomial elements and thereby finding the characteristic equation and transfer functions of a linear time-invariant system in symbolic form.

Also a new approach to the root locus method is shown, using a FORMAC program. The advantages over the conventional root locus method are discussed.

Areas of possible future use of FORMAC in algebraic problems of control theory are discussed.

DD FORM 1 NOV 65 1473 (PAGE 1)

S/N 0101-807-6811

Security Classification

A-31408

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
FORMAC Application						
Mason flow graph						
Algebraic manipulation						
Control theory						
Root locus						
Symbolic system functions						
Linear system analysis						
Flow graph analysis						
Characteristic equation						
Sensitivity analysis						

22 MAY 78

Thesis
M5464 Merzhäuser
c.1
113834
Symbolic algebraic manipulation by digital computer in problems of control theory.